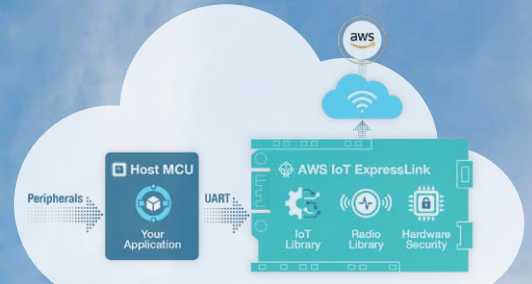
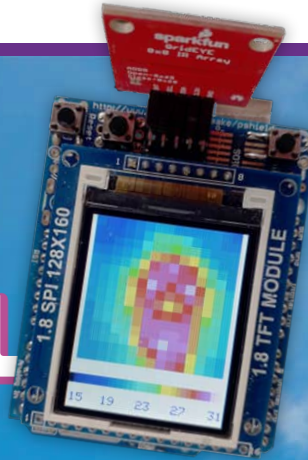


An Arduino UNO-
Based DIY Solution

SMALL THERMAL IMAGING CAMERA

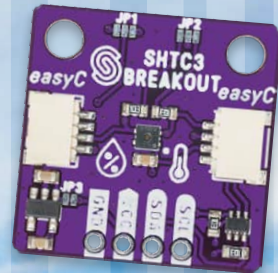
FOCUS ON

IoT & Sensors



AWS for Arduino and Co. (1)

Using AWS IoT
ExpressLink in
Real Life



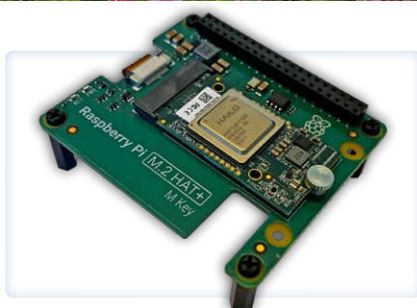
Weather Station Sensors

Which One Should
You Choose?



Universal Garden Logger

A Step Towards
AI Gardening



Raspberry Pi Goes AI
Accelerator with 13 TOPS



ESP32 Energy Meter
Running with Home Assistant



Analog 1-kHz Generator
Sine Waves with Low Distortion



Join the Elektor Community



Take out a
membership!



- ✓ The Elektor web archive from 1974!
- ✓ 8x Elektor Magazine (print)
- ✓ 8x Elektor Magazine (digital)
- ✓ 10% discount in our web shop and exclusive offers
- ✓ Access to more than 5000 Gerber files



Also available

The Digital
membership!



- ✓ The Elektor web archive from 1974!
- ✓ 8x Elektor Magazine (digital)
- ✓ 10% discount in our web shop and exclusive offers
- ✓ Access to more than 5000 Gerber files



www.elektormagazine.com/Member

Volume 50, No. 530
July & August 2024
ISSN 1757-0875

Elektor Magazine is published 8 times a year by
Elektor International Media b.v.
PO Box 11, 6114 ZG Susteren, The Netherlands
Phone: +31 46 4389444
www.elektor.com | www.elektormagazine.com

For all your questions
service@elektor.com

Become a Member
www.elektormagazine.com/membership

Advertising & Sponsoring
Büsra Kas
Tel. +49 (0)241 95509178
busra.kas@elektor.com
www.elektormagazine.com/advertising

Copyright Notice
© Elektor International Media b.v. 2024

The circuits described in this magazine are for domestic and educational use only. All drawings, photographs, printed circuit board layouts, programmed integrated circuits, digital data carriers, and article texts published in our books and magazines (other than third-party advertisements) are copyright Elektor International Media b.v. and may not be reproduced or transmitted in any form or by any means, including photocopying, scanning and recording, in whole or in part without prior written permission from the Publisher. Such written permission must also be obtained before any part of this publication is stored in a retrieval system of any nature. Patent protection may exist in respect of circuits, devices, components etc. described in this magazine. The Publisher does not accept responsibility for failing to identify such patent(s) or other protection. The Publisher disclaims any responsibility for the safe and proper function of reader-assembled projects based upon or from schematics, descriptions or information published in or in relation with Elektor magazine.

Print
Senefelder Misset, Mercuriusstraat 35,
7006 RK Doetinchem, The Netherlands

Distribution
IPS Group, Carl-Zeiss-Straße 5
53340 Meckenheim, Germany
Phone: +49 2225 88010



Jens Nickel

International Editor-in-Chief, Elektor Magazine



Sensing Innovation

When I think of electronics trade fairs, the first shows that come to mind are embedded world and electronica — both are must-attend events for Elektor, and of course we have our own stand there. Incidentally, I can already announce that we have something very special in the pipeline for electronica in November — more about that in the coming issues.

But, it's also the smaller trade fairs that are worth a visit. This week I returned from Sensor+Test, which takes place in Nuremberg, in parallel with PCIM Europe (Power Conversion and Intelligent Motion). With the many small stands, the sometimes highly specialized exhibitors, and not least the accommodation in the somewhat "dusty" exhibition halls 1 and 2, many a visitor will probably feel a little transported back in time at first. You will look in vain here for stands costing millions, with which companies present how they imagine market power based on artificial intelligence in 10 years' time. However, smaller companies in particular are also drivers of innovation. I saw a lot of interesting things at Sensor+Test, for example a handheld device that visualize audio and noise sources. Combined with an image from an optical camera as well as a thermographic image, the result is a new type of tool for machine maintenance.

In addition to Sensor+Test (in line with the focus of this issue), I also visited the PCIM trade fair. You can find a short report in the digital bonus issue, which you can download free of charge from the Elektor website (www.elektormagazine.com/2407-bonus). We are pleased to introduce digital bonus editions for all our magazines. These editions, available as free PDF downloads from the Elektor website, offer additional projects and background articles to complement our regular issues. You may be familiar with this format from our recent Circuit Special or the collaborative guest edition with our friends from Espressif, which has seen over 114,000 downloads!.

You can find these bonus editions on the special theme pages we keep updated year-round. If you have a particular interest in IoT and sensors, visit our IoT and Sensors theme page for ongoing news and exclusive articles throughout the year (www.elektormagazine.com/iot-sensors).

In this edition of Elektor, my colleague, Saad Imtiaz, presents his ESP32-Based Energy Meter — fully working for the first time — in combination with the impressive Home Assistant framework (page 12). IoT fans should definitely also take a look on Tam Hanna's article on page 62 — a step-by-step hands-on article on how to connect small microcontroller boards to the powerful Amazon Web Services.

ElektorLabs Ideas & Projects

The Elektor Labs platform is open to everyone. Post electronics ideas and projects, discuss technical challenges and collaborate with others.

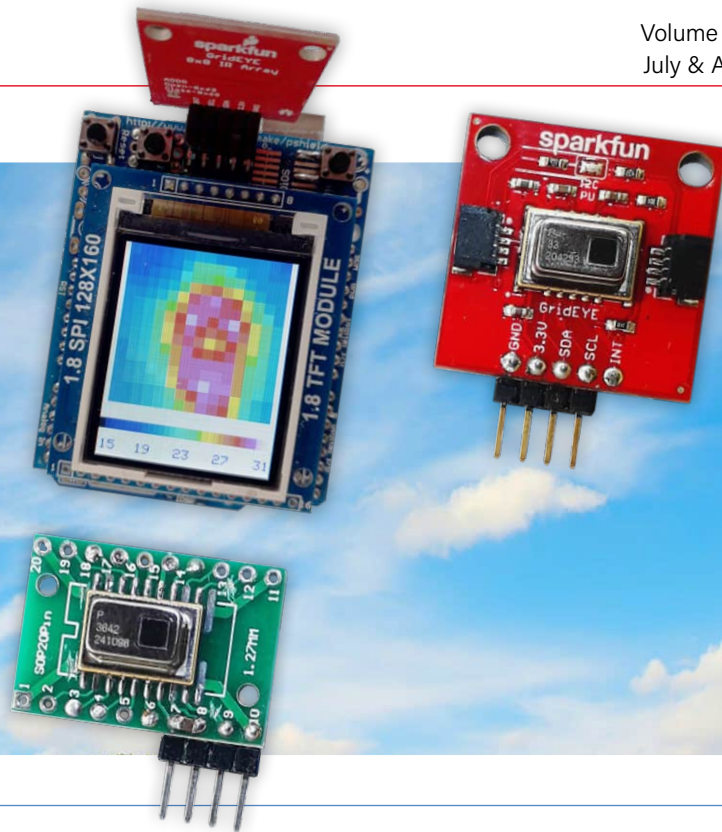
www.elektormagazine.com/labs

The Team

International Editor-in-Chief: Jens Nickel | **Content Director:** C. J. Abate | **International Editorial Staff:** Asma Adhimi, Roberto Armani, Eric Bogers, Jan Buiting, Stuart Cording, Rolf Gerstendorf (RG), Ton Giesberts, Ouafae Hassani, Hedwig Hennekens, Saad Imtiaz, Alina Neacsu, Dr. Thomas Scherer, Jean-Francois Simon, Clemens Valens, Brian Tristram Williams | **Regular Contributors:** David Ashton, Tam Hanna, Ilse Joostens, Prof. Dr. Martin Ossmann, Alfred Rosenkränzer | **Graphic Design & Prepress:** Harmen Heida, Sylvia Sopamena, Patrick Wielders | **Publisher:** Erik Jansen | **Technical questions:** editor@elektor.com

SMALL THERMAL IMAGING CAMERA

An Arduino UNO-
Based DIY Solution



6

Regulars

- 3 **Colophon**
- 18 **2024: An AI Odyssey**
Enhancing Object Detection: Integrating Refined Techniques
- 44 **From Life's Experience**
The Gender Gap
- 80 **Peculiar Parts**
Crystals
- 112 **Starting Out in Electronics...**
...Balances Out

BONUS CONTENT

Check out the free IoT & Sensors
bonus edition of ElektorMag!

- Project: Smart Roller Shutter
- New Products seen at Sensor+Test and PCIM in Nuremberg
- Background: How Do Capacitive Touch Sensors Work?
- Review: HT-03 Thermal Imaging Camera
- And more!



www.elektormagazine.com/2407-bonus

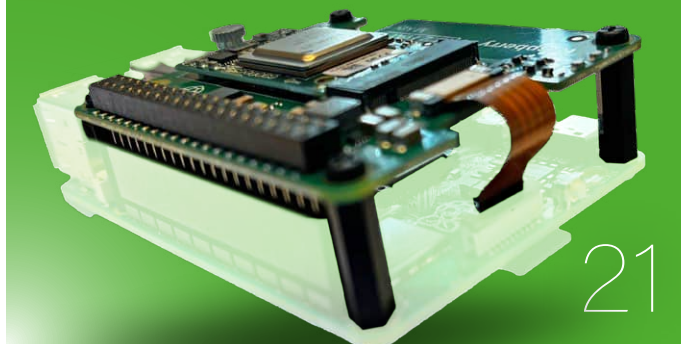
Features

- 21 **Raspberry Pi Goes AI**
New Kit Incorporates M.2 HAT+ With AI Accelerator
- 24 **Weather Station Sensors**
Which One Should You Choose?
- 38 **Elektor Books**
Low-Power Thread Devices Optimized and Scrutinized
- 50 **SparkFun Thing Plus Matter**
A Versatile Matter-Based IoT Development Board
- 62 **AWS for Arduino and Co. (1)**
Using AWS IoT ExpressLink in Real Life
- 94 **Miletus: Using Web Apps Offline**
System and Device Access Included!
- 102 **Interview: From 4G to 5G**
Is It Such an Easy Step?

Projects

- 6 **Small Thermal Imaging Camera**
An Arduino UNO-Based DIY Solution
- 12 **Project Update #3: ESP32-Based Energy Meter**
Integration and Testing With Home Assistant
- 30 **AI-Based Water Meter Reading (1)**
Get Your Old Meter Onto the IoT!
- 34 **A GSM Alarm**
Harnessing GSM Technology for Remote Garage Safety

Raspberry Pi Goes AI Official AI Accelerator Kit Introduced



Weather Station Sensors

Which One Should
You Choose?

- 46 DIY Cloud Chamber**
Making Invisible Radiation Visible
- 70 Airflow Detector Using Arduino Only**
No External Sensors Needed!
- 73 Water Leak Detector**
Connected to Arduino Cloud
- 82 Universal Garden Logger**
A Step Towards AI Gardening
- 89 Analog 1 kHz Generator**
Sine Waves With Low Distortion

Immerse Yourself in IoT & Sensors

Visit Elektor's IoT & Sensors page
for projects, videos, and tutorials!

www.elektormagazine.com/iot-sensors



Industry

- 54 IoT Retrofitting**
Making RS-232 Devices Fit for Industry 4.0
- 57 Enabling IoT With 8-Bit MCUs**
- 60 Technology Drives Sustainability**
Advances Lead to More Efficient Use of Energy

Next Editions

Elektor Magazine Circuit Special Edition

(August & September 2024)

In the tradition of the Summer Circuits Guide, next edition will be extra thick, filled with dozens of DIY-projects, retro circuits, tips and tricks and much more!

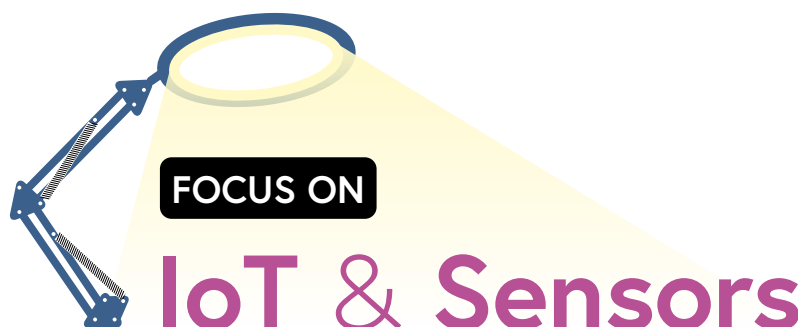
Elektor Magazine Circuit Special 2024 edition will be published around August 14, 2024.

Elektor Magazine September & October 2024

As usual, we'll have an exciting mix of projects, circuits, fundamentals, and tips and tricks for electronics engineers and makers. Our focus will be on Wireless Applications.

- > ESP32 Range Extender
- > LoRa Station with the compact WIO E5 module
- > Expansion Board for ESP32S3 XIAO
- > AWS Cloud for Arduino and Co.: Sending Data
- > RF Probe With LED Bar Graph
- > Configurable Notch Filter

Elektor Magazine's September & October 2024 edition will be published around September 11th. Arrival of printed copies for Elektor Gold members is subject to transport.



Small Thermal Imaging Camera

An Arduino UNO-Based DIY Solution

By Roland Stiglmayr (Germany)

Anyone who has ever needed to detect hot spots on an electronic assembly, measure the temperature of a power component, or detect people in a room has probably wished for a thermal imaging camera. In this article, we present a low-cost DIY solution based on an 8×8 pixel sensor from Panasonic.

An ambitious electronics engineer will certainly wonder whether and how you can build a thermal imaging camera yourself. The answer is: "Yes, you can if you use the right components." The most suitable thermal sensor is the AMG88xx "Grid-EYE" from

Panasonic. Due to its internal signal processing, which completely takes over the calculation of the temperatures of the 64 measuring points, it greatly reduces the load on the external computer. Therefore, a UNO with a 1.8" TFT screen like the one in **Figure 1** is completely sufficient to build a simple thermal imaging camera.

The Infrared Array Sensor

The sensor of the AMG88xx [1] is a chip mounted on a ceramic carrier and manufactured using MEMS technology with 64 infrared-sensitive measuring points (pixels). The pixels are arranged in a square matrix. Next to it is an ASIC with the aforementioned evaluation electronics and a thermistor for recording the reference temperature. The ceramic carrier is covered with a metal cap in which a lens is embedded. The lens is made of silicon so as not to attenuate the radiation in the wavelength range of 5 to 12 μm .

The lens images the target on the sensor field. It can be imagined as if the tip of a square

pyramid were placed on each pixel, and it only receives the radiation directed within this pyramid. Thus, a total of 64 object surfaces arranged in a square are detected, resulting from the arrangement of the pyramids.

Figure 2 shows a simplified and idealized representation of an array with 4×4 measuring points. The field of view (FOV) is the area visible to the array, which is determined exclusively by the vertical and horizontal aperture angles. As the array is a square, the angles are equal and are labeled α . For each object distance a , there is a visible surface within the field of view with the edge length S . The surface in turn consists of 16 individual surfaces with the edge length s_{pix} , each of which is assigned to a specific pixel. The object temperature is only measured accurately if the object completely covers at least one partial area. This requirement can be used to calculate the smallest detectable object size for a specific distance a . In the case of a square matrix with $n \times n$ pixels, the following applies:

$$s_{\text{pix}} = 1/n (2a \tan(\alpha/2))$$

$$A_{\text{pix}} = s_{\text{pix}}^2$$

In **Figure 2**, only the part of the object with the coordinates (2,2) fulfills this requirement. The areas of the field of view with the coordinates (2,1), (1,2), (2,3) are only half covered. The measured temperature change of the associated pixels is therefore only half that of the area (2,2).

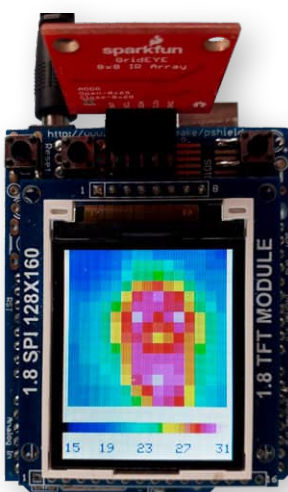


Figure 1: A smart brain obviously generates some heat.

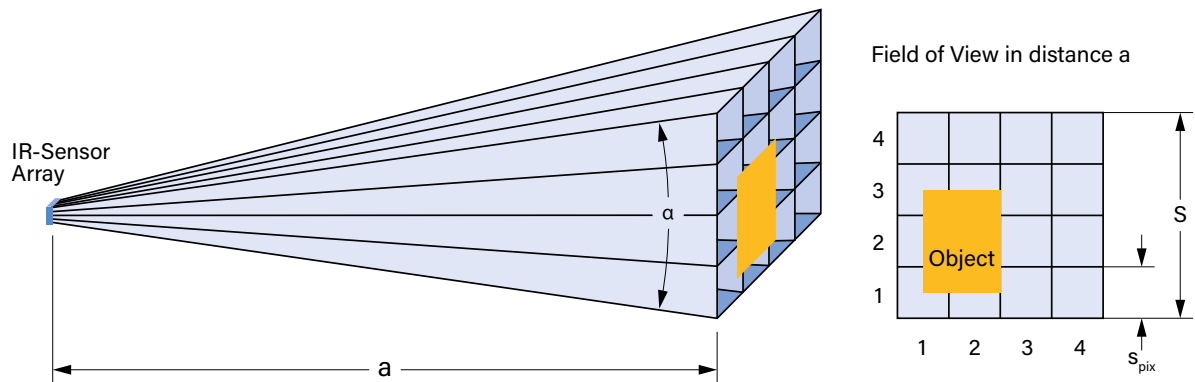


Figure 2: The visible area of a square 4x4 sensor with a field of view α at a distance a .

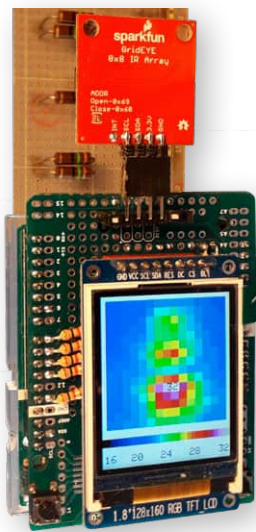


Figure 3: Measuring object at a distance of 50 mm, $\alpha = 60^\circ$.

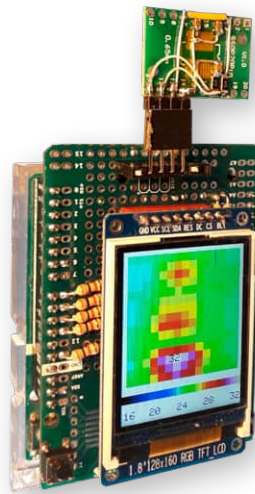


Figure 4: Measuring object at a distance of 100 mm, $\alpha = 30^\circ$.

Put simply, this means that an object must be larger and larger with increasing distance so that the area projected onto a pixel is fully captured. This relationship must always be taken into account in order to avoid incorrect measurements and interpretations. **Figure 3** and **Figure 4** are examples that show the relationship between the field of view and the distance to the measurement object.

The accuracy of the measurement also depends on the emissivity ϵ of the object. It results from the composition of the surface and determines the ability to emit thermal radiation. The Grid-EYE sensor assumes a fixed emissivity of 0.93 in its internal calculation of the temperature.

Thermopile

Each individual measuring point of the sensor field is what is known as a thermopile [2], which consists of many thermocouples

connected electrically in series and thermally in parallel. The ceramic carrier provides the reference temperature, the irradiated surface gives the temperature of the thermopile to

be measured. Thermocouples generate a voltage that is proportional to the temperature difference. To determine the absolute temperature, the reference temperature must be measured very precisely and added to the temperature difference. The characteristic curve of a thermocouple is not linear, but is a function of the reference temperature and the measured voltage. The measured values are linearized using curves stored in the AMG's ROM. Sample scattering of the individual thermopiles is taken into account by individual, internally stored calibration data. It is easy to imagine how time-consuming or computationally intensive the linearization of the characteristic curve would be if the internal ASIC did not fortunately take care of this task. The ASIC provides the linearized and perfectly processed measured values in degrees Celsius via its I²C interface. Different sensor types are available depending on the application. **Table 1** provides an overview.

Table 1: Typical values of the Infrared Array Sensors of the AMG88xx series.

Module	VDD	Measuring range	Total angle of detection	Angle of detection of a pixel	Object size in relation to one pixel at a distance of 0.3 m
AMG8832	3.3 V	-20°C ... 100°C	60°	7.5°	4 cm
AMG8833	3.3 V	0°C ... 80°C	60°	7.5°	4 cm
AMG8834	3.3 V	-20°C ... 100°C	60°	7.5°	4 cm
AMG8853	5 V	0°C ... 80°C	60°	7.5°	4 cm
AMG8854	5 V	-20°C ... 100°C	60°	7.5°	4 cm
AMG883642	3.3 V	-20°C ... 100°C	32° ... 34°	4°	2 cm
AMG883543	3.3 V	0°C ... 80°C	90°	11.25°	6 cm

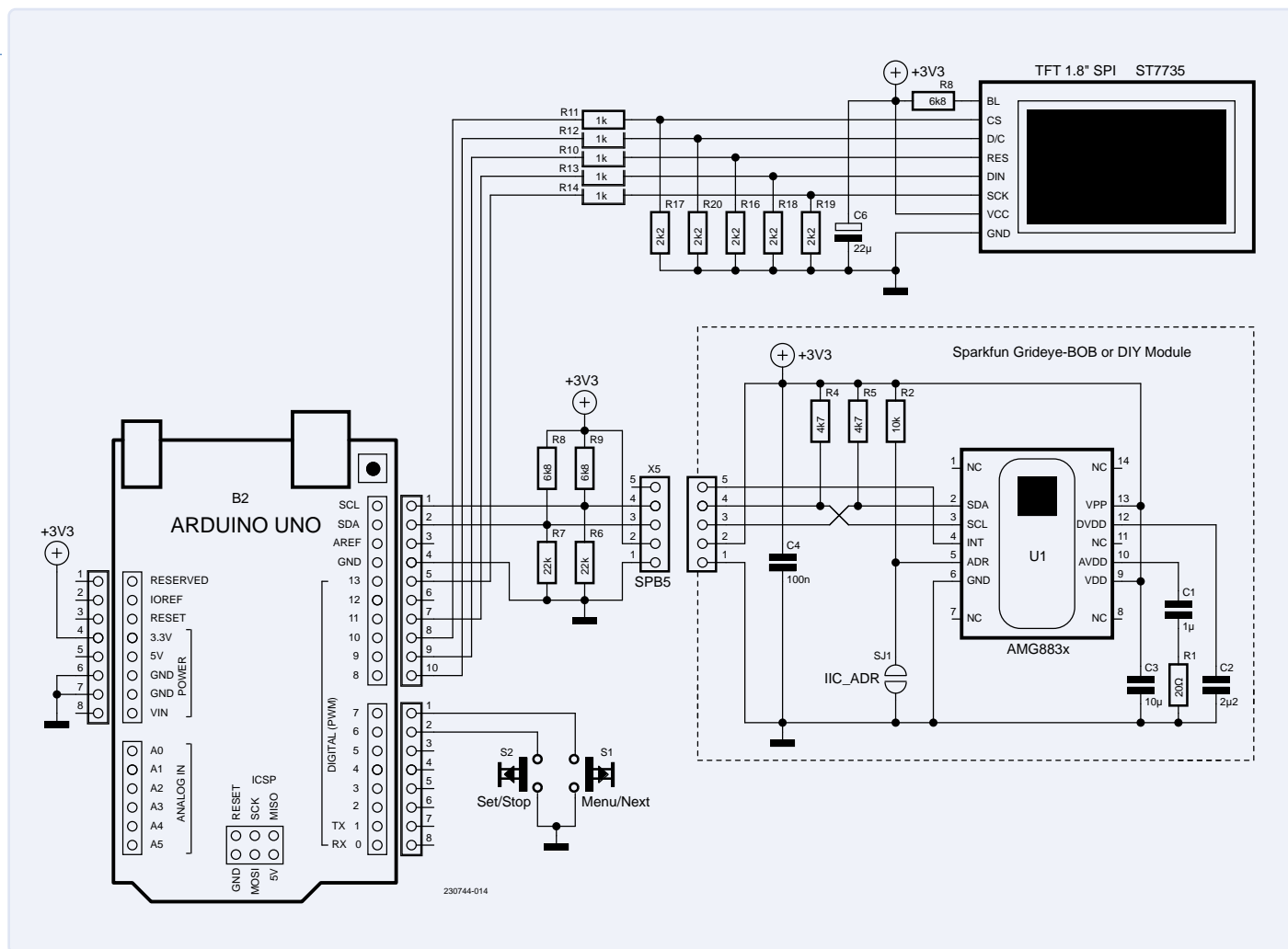


Figure 5: Circuit diagram of the camera.

The Camera Structure

The entire circuit of the thermal imaging camera consists of an Arduino UNO, a shield with a small TFT color display, two buttons, and the camera module. **Figure 5** shows the complete circuit diagram of the camera.

Camera Module

If you decide to use a fully assembled break-out board [4] (**Figure 6**) with an AMG8833, you are already done. You then only have to consider whether the cabling should be realized with Qwiic or pin headers.

However, you can also build the module yourself relatively easily, as the sensor can be soldered without any problems using a fine soldering iron. The advantage of the self-built version is that you can choose a suitable sensor. For example, the AMG883642 [3] is very interesting for many applications due to its small field of view and wide temperature range.

The author has soldered the sensor onto an SSOP20 to DIL adapter module (1.27 mm by

2.54 mm) and also included the few other components on the module in SMD (**Figure 7**). The I²C device address is irrelevant, as the software determines the correct address at startup, so that the ADDR pin can be connected to GND.

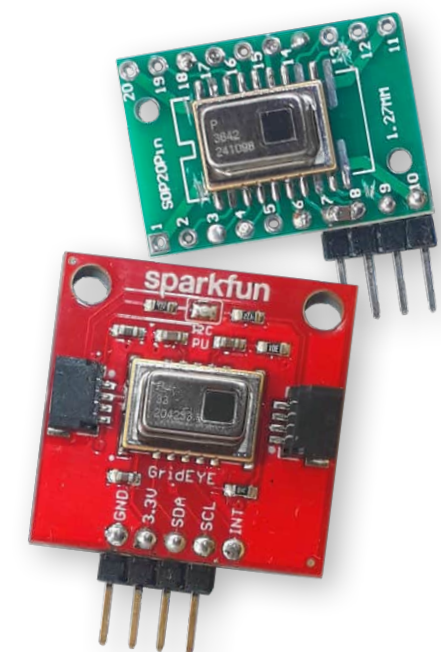


Figure 6: DIY sensor module next to the Sparkfun BoB.

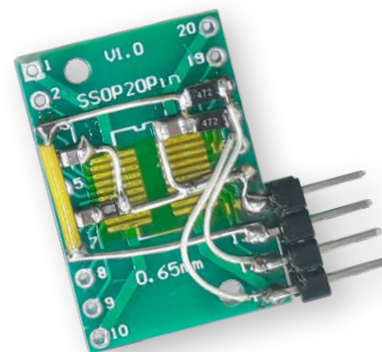


Figure 7: DIY sensor module, rear side.

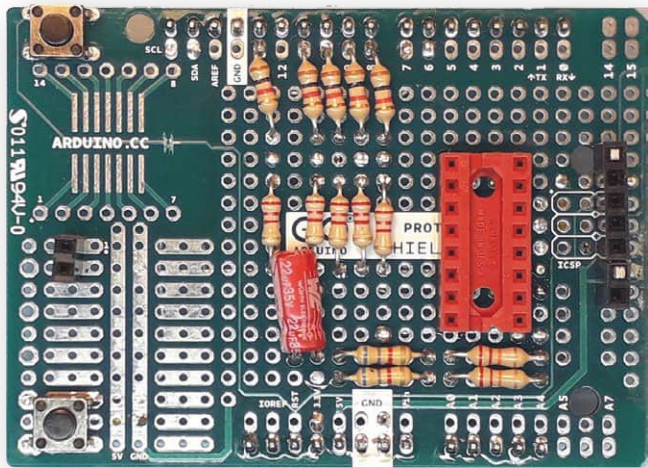


Figure 8: Self-made shield, assembly side.

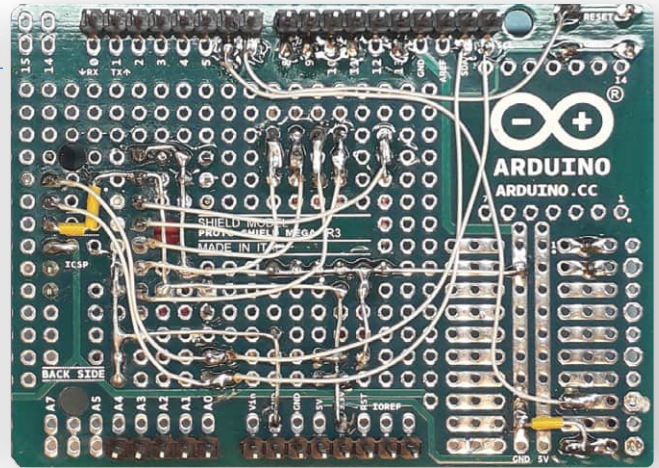


Figure 9: Self-made shield, wiring side.

TFT Shield

Since the circuit in Figure 5 is so simple, it can easily be built on an Arduino Mega Proto Shield. There is enough space there to use wired components (**Figure 8**, **Figure 9**). The TFT display with a diagonal of 1.8 inches and 128×160 pixels must have an ST7735 controller and should be able to operate with 3.3 V. A cheap product made in China is perfectly adequate. But take care: These boards have different pin assignments. This is not a problem, but you have to pay attention to it. A DIL socket is a very suitable connector for the display.

Power Supply and Buses

The camera module is powered by the UNO's 3.3 V supply. For this reason, the pull-up resistors of the I²C lines are connected to 3.3 V. In addition to the resistors on the camera module, there are also two pull-ups to 3.3 V on the author's shield and two pull-downs to compensate for the internal current of 140 µA from the ports. Although the UNO runs at 5 V, it reads the 3.3 V signals without any errors.

The TFT display is controlled via the SPI interface. Its ST7735 controller only accepts 3.3 V signals at its interface. The 5 V port signals of the UNO are therefore reduced to 3.3 V using the voltage dividers R10 and R14 as well as R16 and R20. As the SPI clock is 10 MHz, the resistance of the resistors must not be too high, otherwise the parasitic influences would be too disruptive. The specified values represent a good compromise between current consumption and distortion-free signal transmission. The placement of the two buttons can be seen in **Figure 10**.

Setup

The first step is to install the two libraries, *Adafruit_ST7735_and_ST7789_Library*

and *Adafruit_GFX_Library* using the library manager of the Arduino IDE. Then compile the sketch *Grideye_V2x.ino* from the Elektor project page [5] and load the program into the UNO.

With the power supply switched off, plug in the shield, still without the camera module, and power the UNO. The TFT should now light up and display the error message "no valid device found". Now check the polarity of the supply voltage at the connector to the camera module. The I²C signals can be measured with an oscilloscope. Then switch off again, connect the camera module to the shield, and put the module back into operation.

The result is (hopefully) a meaningful image. Depending on the version of the ST7735 controller, the position and the colors may be incorrect. This can be corrected in the program. The sketch offers four different initialization routines for this purpose. Four values `TFT_TYPE_n` are defined for this purpose, one of which must be assigned to the constant `TFT_TYPE`. With the correct value, the image is fitted at the top, left and right. If the red and blue colors are reversed, the value `TFT_CHANGE_COLOR` must be set to true. If the image is upside-down, assign the value `02` to `TFT_ORIENTATION`. Finally, the temperature range of the sensor used is selected by setting the value `AMG88x3` to `true` or `false`.

```
#define AMG88x3 false
//select sensor type, true if AMG88x3
#define PERMANENT_AUTO false
//if true autoranging always active
#define T_OFFSET 0
//offset for correcting T [°C]

#define TFT_ORIENTATION 00
```

```
//portrait format
#define TFT_ORIENTATION 02
//portrait format overhead

#define TFT_TYPE_1 00
//module type 1 (w/o SD)
#define TFT_TYPE_2 01
//module type 2
#define TFT_TYPE_3 02
//module type 3 (with SD)
#define TFT_TYPE_4 03
//module type 4
const byte TFT_TYPE= TFT_TYPE_1;
//set used module type

#define TFT_CHANGE_COLOR true
//change red-blue depending on tft
```

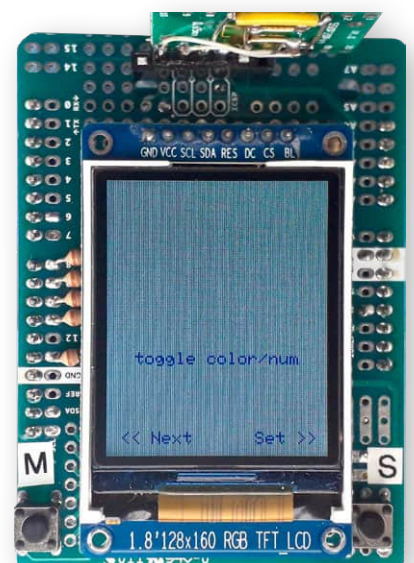


Figure 10: Button positioning, switching between graphical and numerical display.



Listing 1: The read_GridEYE Function.

```
if (acquire==true)
    (read_GridEYE (&lowestT, &highestT,                //read all pixels and
                  &highestTpix));                     //get lowest and highest T

...

void read_GridEYE (int *pl, int *ph, int *phnum)
{
    *ph= -512; *pl= 1024;                               //init values highest, lowest
    int T;                                              //temporary
    byte pixnum;

    // read data of 64 pixels
    // -----

    for(byte y = 0; y <8; y++)                          //line address
    {
        for(byte x=0; x<8; x++)                        //column address
        {
            pixnum= (y+56)-(x*8);                      //due to orientation, start with pixel 56
            T= GridEye_getT (pixnum);                  //read this pixel
            pixel_Table[y*2][x*2]= T;                  //T to each second column, line position

            if (T>*ph)                                  //find highest temperature
            { *ph=T;
              *phnum= y*256 + x;                        //save position of this pixel
            }
            if (T<*pl) *pl=T;                          //find lowest temperature
        }
    }

    // interpolate between the horizontal pixels of each line
    // -----

    for (byte y=0; y<8; y++)                            //line address counter
    {
        for (byte x=0; x<7; x++)                        //column address counter
        {
            T= (pixel_Table [2*y][2*x] + pixel_Table [2*y][(2*x)+2])/2;
            pixel_Table [2*y][(2*x)+1]= T;
        }
    }

    // interpolate between the vertical pixels of each column
    // -----

    for (byte x=0; x<15; x++)                            //column address counter
    {
        for (byte y=0; y<7; y++)                        //line address counter
        {
            T= (pixel_Table [2*y][x] + pixel_Table [(2*y)+2][x])/2;
            pixel_Table [(2*y)+1][x]= T;
        }
    }
}
```


Software

The sketch was created in the Arduino programming environment. No external library is used for the Grid-EYE sensor; all the necessary functions are implemented directly in the sketch. This makes it easy to implement changes or extensions according to your own ideas.

The main program calls the `read_GridEYE` measurement routine (**Listing 1**) cyclically (i.e., the process is deterministic). The measurement routine reads the 64 temperatures via the `GridEye_getT` function (the lowest and highest temperatures are also determined during reading and saved together with the position of the highest temperature).

The temperatures have a resolution of 0.25 K. To avoid fractional numbers, they are multiplied by a factor of 4 by the sensor. The sensor then supplies the values with 12 bits, where the 12th bit is the sign. The read routine converts this into an int type, i.e. 16 bits in two's complement.

The sensor values are written to the two-dimensional array `pixel_Table`. This array stores 15×15 int values and can therefore also hold the interpolated values. After reading in the sensor values, the routine generates additional values between two neighboring row values by interpolation. This interpolation is then also carried out column by column, including the already interpolated values. The resulting 15×15 = 225 values are now available in the `pixel_Table` array.

The `showT_as_Color` function displays the 225 values in false colors. The colors are assigned to the values via the LUT `color_Scale`. For good differentiation, the resolution is limited to 32 colors.

The measured values must therefore be scaled so that they fit into a range of

32 values. This is done by the `do_Auto_Ranging` function. Depending on the lowest measured temperature, the function selects the lowest value of the measuring range from six predefined values and passes it to the `Tcoloffset` variable. The difference between the highest measured temperature and `Tcoloffset` is used to determine the resolution `Tcolrange` of the measuring range from five predefined values. The measured values are transformed into a value range of 32 by subtracting `Tcoloffset` and multiplying by `Tcolrange` so that they can be represented by 32 colors.

The numerical display of the measured values is handled by `showT_as_Numeric`, while the serial output is handled by the `Serial_send_T` function. The `Set_menu` routine provides a selection of various settings and applies them when activated. The settings are saved to the EEPROM and restored at each restart.

Operation

The **Menu** button takes you to the settings. By pressing this button repeatedly, you can scroll through the selectable items. The currently active settings are displayed in green. Press the **Set** button to apply the displayed entry. The first parameter is **Auto Ranging**, which usually achieves an optimal display. With **toggle color/num** (see Figure 10), the results can be displayed numerically or in false colors.

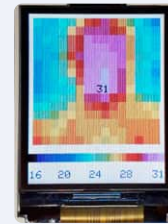
If the **Set** button is pressed during the normal measurement process, the measurement is stopped and frozen. The display can then be edited in the menu without changing the measured values. Press again to continue the measurement.

In the false color display, the temperature of the warmest point is displayed as a numerical value in the graphic. This feature can be suppressed when restarting by holding down the **Menu** button during the reset. The results

are output continuously via the serial interface with a resolution of 0.25 K.

In conclusion, one can only watch in amazement what such a small ATmega processor is capable of! ◀

Translated by Jörg Starkmuth — 230744-01



About the Author

Roland Stiglmayr studied information technology in the 1970s and has over 40 years of experience in research and development. His work has focused on the development of computer mainframes, fiber optic-based data transmission systems, RRHs for mobile communications, and wireless energy transmission systems. Today, he works in an advisory capacity. He is particularly committed to the transfer of knowledge.

Questions or Comments?

Do you have questions or comments about this project? Email the author at 1134-715@online.de or contact Elektor at editor@elektor.com.



Related Product

> **SparkFun Grid-EYE Infrared Array Breakout – AMG8833**
www.elektor.com/19605



WEB LINKS

- [1] Grid-EYE documentation: <https://industrial.panasonic.com/ww/products/pt/grid-eye>
- [2] Thermopile (Wikipedia): <https://en.wikipedia.org/wiki/Thermopile>
- [3] AMG883462 data sheet: <https://industrial.panasonic.com/cdbs/www-data/pdf/ADI8000/ast-ind-139049.pdf>
- [4] Grid-EYE-BoB: <https://sparkfun.com/products/14607>
- [5] Project page: <https://elektormagazine.com/230744-01>



Project Update #3: ESP32-Based Energy Meter

Integration and Testing
with Home Assistant

By Saad Imtiaz (Elektor)

In the previous project update, you learned about enhancements to the ESP32 Energy Meter's schematic design and PCB. This article focuses on the practical implementation and integration of the new version. It provides a step-by-step guide on setting up the meter with ESPHome and Home Assistant for effective energy monitoring. Furthermore, we deal with the device's calibration.

In our previous article [1], we focused on enhancements to the schematic design and PCB of the ESP32 Energy Meter, with improvements in modularity and safety features. Before we get into the next project update, let's have a brief overview.

In the most recent advancements of the ESP32 Energy Meter project, we upgraded to the ESP32-S3 microcontroller, introducing enhanced processing power and broader functionality. The new design slimmed down the PCB and incorporated a transformer-based power system,

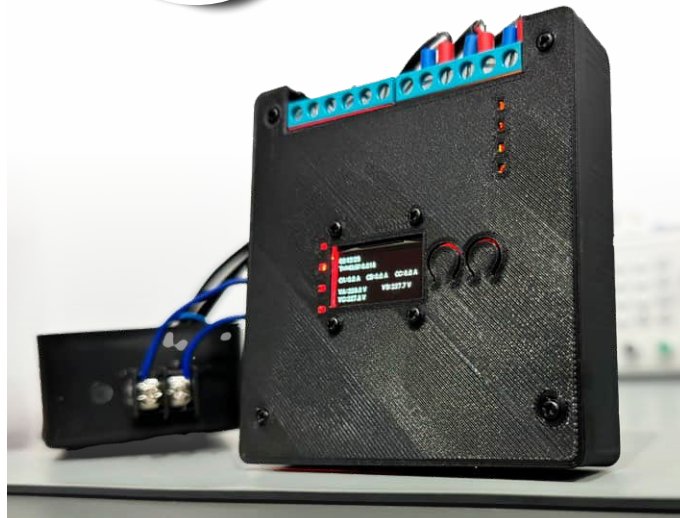


Figure 1: The assembled ESP32 Energy Meter with its OLED display and live status indicators.

using a 230-V-to-12-V Step Down Transformer for voltage sampling and running of the system. This significantly improves safety while maintaining flexibility for both single- and three-phase installations.

Other improvements included the integration of a more efficient AP63203WU-7 buck converter, making the PCB more modular, calibration of the current transformer sampling circuit, and more. This not only optimizes the energy meter's performance and functionality, but also reduces both cost and size.

In this article, we will discuss the steps taken to get this energy meter up and running and the journey it took from the lab bench to the circuit breaker box. Furthermore, we will also discuss how to set it up, calibrate it, and finally, how to integrate it with Home Assistant with ESP Home to show and monitor the collected data from the energy meter. In **Figure 1**, a vivid snapshot of the ESP32 Energy Meter project in action is captured, encased in a 3D-printed enclosure with an OLED display. The image highlights live status indicators that seamlessly track and display real-time power consumption.

Assembly

The new PCB was designed to be more compact and straightforward to solder and the layout had adequate spacing for each component which accommodated the soldering process of it. To facilitate project replication and modifications by enthusiasts and professionals alike, the complete Bill of Materials (BOM) in Mouser format, and the production files are shared on the Elektor's Lab GitHub repository [2].

For the voltage and current sampling connections, screw type terminal blocks by CUI Devices were used, the quality of these terminal blocks were much better than the cheap blue colored terminal blocks seen on most sensor modules. As we are dealing with AC Voltages and energy metering, it is vital to have secure and reliable connections.

Noise reduction is a critical aspect of the PCB design, addressed by integrating both electrolytic and ceramic capacitors around the ATM90E32S energy metering chip. This arrangement helps to filter out both low- and high-frequency noise, ensuring more accurate and stable energy measurement. The board is depicted in **Figure 2**.

As mentioned earlier, we decided to use a step-down transformer for voltage sampling and the main source of powering the entire system. Finding such a transformer is easy and cheap, but most of these step down transformers take up a lot of space when used in a custom enclosure, as shown in **Figure 3**. So, it is best that DIN Rail Bell transformers are used in this case to make the setup more clean and safe; such transformers can easily be found online. Moreover, the accuracy of voltage measurements depends on the characteristics of the transformers, including their voltage ratio accuracy, phase shift, and linearity.

You might have noticed in the images that there is only one transformer attached to the energy meter. As the energy meter was configured to be used in Single Phase mode, by sorting the jumper JP8 on the back side of the PCB, as seen in **Figure 4**. To operate the energy meter in three-phase mode or to sample voltage from each phase in a three-phase system using three step-down transformers, you must connect the primary sides of three transformers to the respective phases (L1, L2, L3). On the secondary side, connect one end of each transformer's winding to a common neutral point, forming a star (Y) configuration. The free ends of the secondary windings (V1, V2, V3) will then provide the voltage outputs (UA, UB and UC) on the PCB for each phase. Key considerations include ensuring the transformers are properly rated for the system's voltage and current, maintaining strict isolation between primary and secondary circuits for safety, and securing a stable and well-balanced neutral connection to prevent measurement inaccuracies.

Setting up with ESPHome and Home Assistant

As part of the development plan, a specific firmware is being created to leverage the capabilities of the energy metering chip and the advanced AI features of the ESP32-S3. Although developing such tailored firmware requires a significant amount of time and is still under-way, this does not restrict the usability of the energy meter. The device can be fully functional with existing platforms like Home Assistant, providing an immediate solution for energy monitoring. Therefore, in this article, the focus is on the integration of the ESP32 Energy Meter with Home Assistant [3] and ESPHome [4]. This section will guide you through setting up the energy meter within the Home Assistant environment to utilize its complete functionality.

To set up the ESP32 Energy Meter with the ESPHome firmware and integrate it into Home Assistant, you should start by installing Home Assistant. (See a comprehensive guide in an article by my colleague Clemens Valens [5].) After that, add the ESPHome integration from the **Add-on Store**; then create a new project in ESPHome for your ESP32 device. This automatically generates a basic YAML configuration file (such a YAML file specifies a distinct ESPHome project with all the sensors used and many other options). You have to download this default file, before taking the next steps.

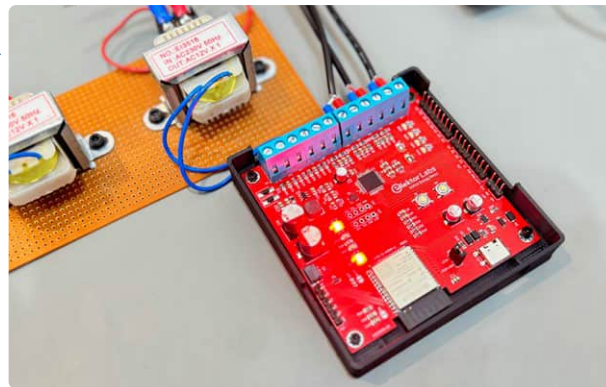


Figure 2: PCB of the fully assembled ESP32 Energy Meter.



Figure 3: 220 V to 12 V Step-down transformer setup within a custom enclosure.

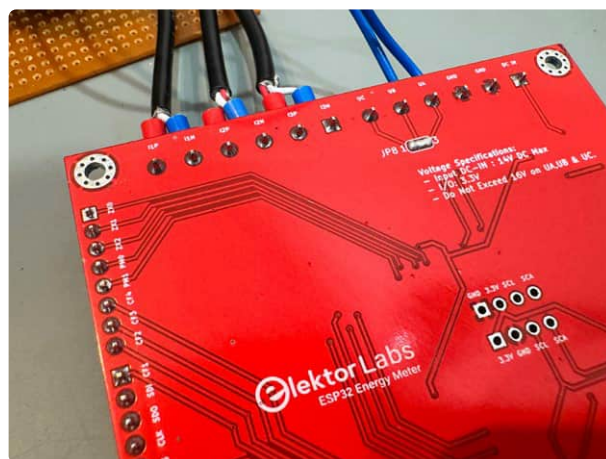


Figure 4: Single-phase jumper configuration on the ESP32 Energy Meter's PCB.

Connect your ESP32 to your computer and select the correct COM Port for the ESP32-S3. In the ESPHome dashboard, click *Install* and choose the *.bin* file to flash the firmware. Once the firmware is successfully uploaded, and your ESP32 Energy Meter is recognized by Home Assistant, proceed to edit the initial YAML configuration. To do this, open the ESPHome dashboard in Home Assistant, find your device, and click on the *Edit* option on the energy meter's card. Replace the existing configuration with the YAML content provided in the GitHub repository [2]. Make sure to properly configure your API, OTA, and WiFi credentials in this new YAML setup.

Install the new configuration wirelessly onto your ESP32 Energy Meter. Once this is done, the device will be active and connected. To display the energy meter data on your Home Assistant dashboard, simply assign the ESPHome device to a specific area in Home Assistant. This helps organize your dashboard by grouping devices according to their physical or logical location in your home. For a visual representation of what you can achieve, refer to the **Figure 5**, which displays the energy meter data on the Home Assistant dashboard.

Integrating the ESP32 Energy Meter with Home Assistant not only simplifies the process of monitoring energy usage but also unlocks a suite of powerful features provided by the platform. Home Assistant offers an intuitive interface for real-time data visualization, control automation, and seamless integration with other smart devices in your home. This integration allows for the creation of detailed history graphs and analytics within Home Assistant, providing an in-depth look at power consumption patterns over time, as shown in **Figure 6**. These insights enable users to make informed decisions about their energy use, identify potential savings, and optimize their home's energy efficiency.

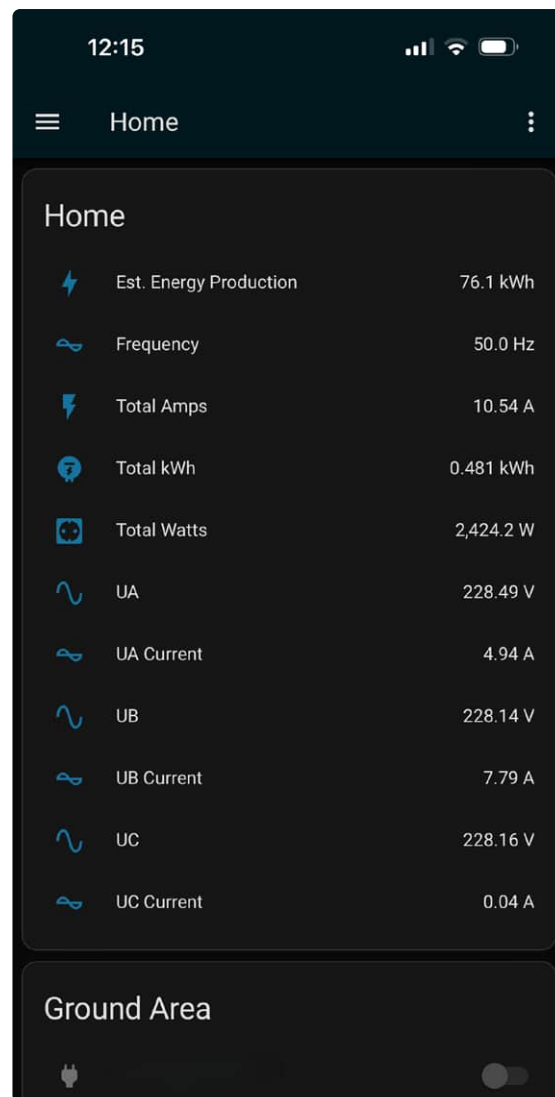


Figure 5: Home Assistant dashboard displaying real-time energy data from the ESP32 Energy Meter.



Figure 6: Detailed history graphs of energy consumption in Home Assistant.

By following the setup process described above, users can take full advantage of these capabilities, turning the ESP32 Energy Meter into a central component of their smart home ecosystem. This integration not only enhances the functionality of the energy meter but also enriches the overall smart home experience with comprehensive energy monitoring and management tools.

YAML Configuration

The provided YAML configuration sets up the ESP32 Energy Meter with ESPHome, enabling the monitoring of essential electrical parameters like voltage, current, power across all three phases. It leverages the capabilities of the ATM90E32 sensor, with detailed definitions for SPI communication and individual sensors for each phase. This setup not only measures but also calculates total consumption metrics, integrating a daily energy counter for kWh and an OLED display for real-time data visualization. These configurations are made according to the instructions on the ESPHome page for the ATM90E32 sensor [6].

For ensuring the accuracy of the data reported by the ESP32 Energy Meter, calibration is a crucial step. The specifics of how to adjust the gain settings for current transformers and voltage inputs will be detailed in the upcoming section.

Test Setup and Calibration

The test setup employed a multi-step heat and speed hair dryer as the load, covering a range from 0.7 A to 8 A. The power cord of an extension power strip was stripped to allow placement of the split coil transformers on the live or neutral wire, facilitating direct monitoring under various conditions as shown in **Figure 7**.

I conducted the current and voltage calibration of the ESP32 Energy Meter using my UT201+ multimeter. This offers a resolution of 0.001 A and an accuracy specification of $\pm 4\%$ +10 digits for current and a resolution of 0.001 V with accuracy of $\pm 1\%$ +5 digits for voltage.



Figure 7: Setup for testing and calibrating the ESP32 Energy Meter using a variable load.



Figure 8: Clamp meter setup for calibration.

This level of precision is adequate for most projects, but is slightly less precise than professional-grade meters.

During the calibration, a comparison of current readings was observed: the clamp meter reading was 1.692 A shown in **Figure 8**, while the readings calculated by the energy meter displayed 1.70 to 1.73 A after calibration, as shown in **Figure 9**. Given the specifications of the UT201+ and the SCT-013-000, a Class 1 split core transformer which guarantees an accuracy within 1% of the actual value, this small discrepancy falls within the expected error margin. However, for even greater accuracy, a more precise clamp meter could be used.

To fine-tune the ESP32 Energy Meter's accuracy further, adjustments were made to the gain settings for both voltage and current measurements. For voltage, the sensor was calibrated using the formula:

$$\text{New gain_voltage} = (\text{your voltage reading} / \text{ESPHome voltage reading}) * \text{existing gain_voltage value}$$

Similarly, for current adjustments:

$$\text{New gain_ct} = (\text{your current reading} / \text{ESPHome current reading}) * \text{existing gain_ct value}$$

These new gain values were then updated in the ESPHome YAML configuration file, followed by recompiling and uploading the firmware. This process can be repeated as necessary to ensure optimal accuracy. These calibrated values help refine the measurements and are crucial for accurate reporting and analysis in any energy monitoring setup.

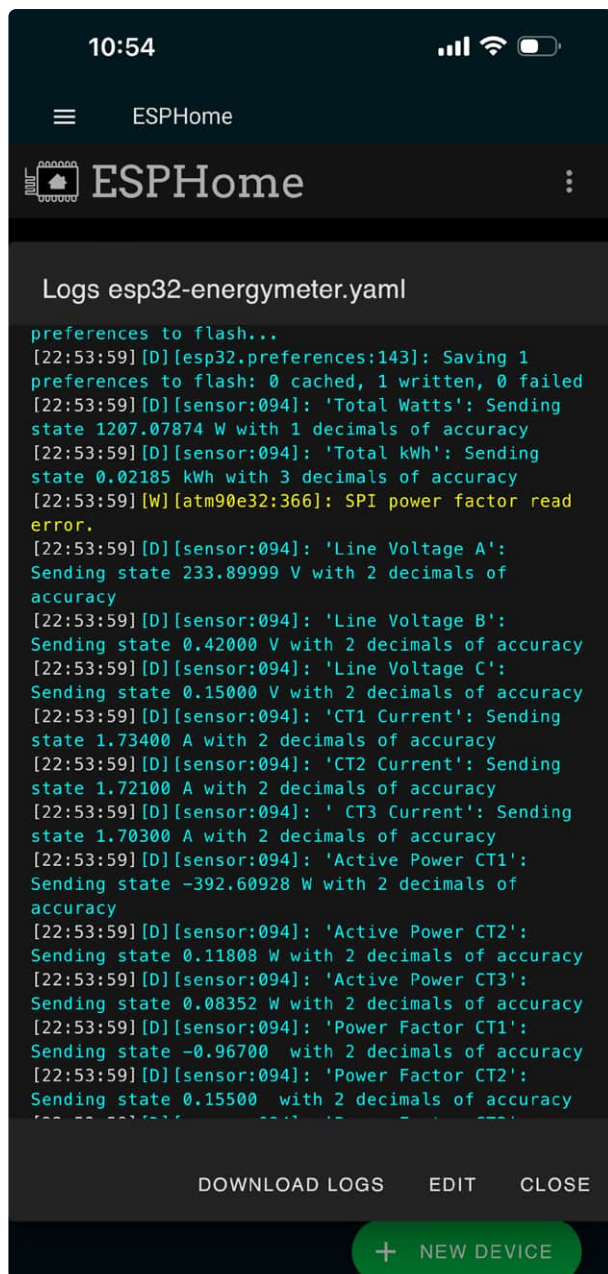


Figure 9: Final calibration results showing improved measurement accuracy.



WARNING: Working inside a circuit breaker box carries inherent risks, including the potential for electrical shock or fire. It is vital to turn off all power before starting the installation. In most countries, this work may only be carried out by a qualified electrician!

Breaker Box Installation for the ESP32 Meter

Installing the ESP32 Energy Meter into my circuit breaker box proved to be a manageable task that required meticulous attention to detail to ensure both safety and functionality. I started by selecting a circuit with the lowest amp limit. This choice was strategic, as it provided a safety buffer; the circuit breaker would trip in the event of any unexpected surges or transformer failures, thus protecting the system.

Using split core current transformers was particularly beneficial due to their ease of installation. These transformers can be quickly clamped onto any load, but it was crucial to pay attention to the direction of current flow to guarantee accurate readings. It's important to note that if the direction of the current and the orientation of the current transformer are not aligned correctly, the power readings will appear negative, which indicates incorrect installation.

For a visual demonstration of the ESP32 Energy Meter in action within the circuit breaker panel, refer to **Figure 10**. This image shows the energy meter displaying real-time current, voltage measurements, and the corresponding load in kilowatts, illustrating its functionality in a live setting.



Figure 10: ESP32 Energy Meter installed in a circuit breaker panel, monitoring real-time power consumption.

Development and Prospects

While the current software configuration runs on ESPHome, there is ongoing development to expand the capabilities of the ESP32 Energy Meter. The project is looking forward to being integrated with a new firmware specifically designed to harness the full potential of the ESP32-S3 chip. This future firmware is expected to include advanced features such as detailed energy analytics and potentially groundbreaking AI/ML functionalities that could predict energy usage patterns and identify the device according to its load footprint.

Although the core design and operational aspects of the project have been completed, the development of these sophisticated features is a complex and time-consuming endeavor. I am excited about the possibilities and committed to pushing the boundaries of what this energy meter can achieve.

The ESP32 Energy Meter project is continuously evolving, incorporating more features with each update. Community members who are interested in the upcoming AI and ML functionalities, or those who would like to contribute to the development, are encouraged to get involved. Collaboration will help accelerate progress and result in a more robust and feature-rich energy monitoring solution. Keep an eye out for further advancements as the project aims to refine and elevate this versatile energy management tool to new heights. ◀

240244-01

Questions or Comments?

If you have questions about this article, feel free to email the author at saad.imtiaz@elektor.com or the Elektor editorial team at editor@elektor.com.



About the Author

Saad Imtiaz (Senior Engineer, Elektor) is a mechatronics engineer with experience in embedded systems, mechatronic systems, and product development. He has collaborated with numerous companies, ranging from startups to enterprises globally, on prototyping and development. Saad has also spent time in the aviation industry and has led a technology startup company. At Elektor, he drives project development in both software and hardware.



Related Products

> **PeakTech 4350 Clamp Meter**
www.elektor.com/18161

> **Siglent SDM3045X Multimeter**
www.elektor.com/17892



WEB LINKS

- [1] Saad Imtiaz, "Project Update #2: ESP32-Based Energy Meter", Elektor 5-6/2024 : <https://www.elektormagazine.com/magazine/elektor-341/62892>
- [2] ESP32 Energy Meter Github Repository: <https://github.com/ElektorLabs/esp32-energymeter>
- [3] Home Assistant: <https://home-assistant.io/>
- [4] ESPHome: <http://esphome.io>
- [5] Clemens Valens, "Home Automation Made Easy", Elektor Magazine 9-10/2020 : <https://www.elektormagazine.com/200019-01>
- [6] ATM90E32 Power Sensor: <https://esphome.io/components/sensor/atm90e32.html>

2024 An AI Odyssey

Enhancing Object Detection: Integrating Refined Techniques

By Brian Tristram Williams (Elektor)

After successfully deploying basic object detection on a headless Raspberry Pi, our journey continues with a focus on refining this technology. This installment covers technical enhancements to improve accuracy and efficiency, alongside integrating supplementary sensors to boost functionality under diverse environmental conditions.



Figure 1: PAL and NTSC video used an interlaced method of drawing scanlines, and this affected full frames when there was movement.

After running into trouble trying to detect objects and text on old archival videotapes, I had to look at ways of improving the detection algorithms. In my case, the fact that the video I was working on is interlaced and low resolution is likely to have been a confounding factor. The differences between alternate fields are likely to confuse the detection model that's set up to analyze one frame (two fields) at a time (**Figure 1**). I have to look at deinterlacing the captured old-school video before running it through the workflow.

Tweaking It Up

However, my niche challenges aside, that doesn't mean we can't improve the detection a bit with some refinements. This is what I found helped when working with non-interlaced digital video coming from the Raspberry Pi Camera Module 3:

Dynamic Thresholding: To counteract varying lighting and distances, I implemented an adaptive confidence threshold. This approach adjusts the detection confidence levels in real-time based on the average brightness detected by the camera, using a simple luminosity algorithm:

```
def adjust_threshold(lux):
    base_threshold = 0.5 # base confidence level
    if lux < 50: # low light conditions
        return base_threshold - 0.1
    elif lux > 500: # very bright conditions
```

```
        return base_threshold + 0.1
    return base_threshold
```

Lux Sensor Integration: Speaking of lighting, while I know that the camera itself is a luminosity sensor of sorts, having torn my hair out trying to adjust the plethora of Camera Module parameters to get my pictures and videos "just right" in the past, I know that the brightness of the output video signal's relation to the actual brightness outside will be tenuous — that's just one more confounding variable whose frustration I don't want to deal with, so I opted to use a sensor for the purpose.

I selected a TLS2561 lux sensor (**Figure 2**) to feed real-time data to the Raspberry Pi, which then dynamically adjusts camera exposure and processing parameters. This ensures optimal image quality for the detection algorithms under varying lighting:

```
lux = read_lux_sensor()
camera.set_exposure(calculate_exposure(lux))
```

Again, I didn't have to write those functions myself, as there's a great *Adafruit-TSL2561* library available [1]. Installation is simple from the terminal:

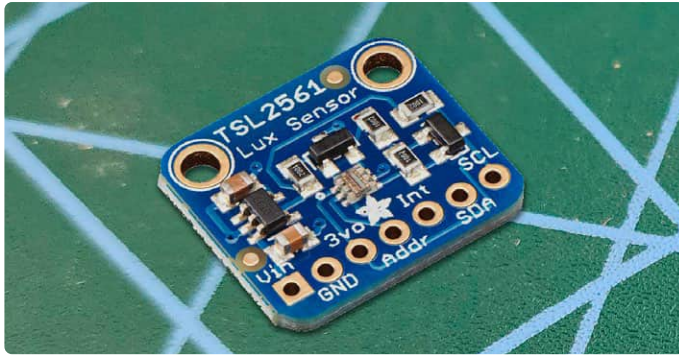


Figure 2: The TSL2561 luminosity sensor on an Adafruit module.

```
sudo apt-get update
sudo apt-get install python3-smbus
pip3 install Adafruit-TSL2561
```

Then with this Python code, you can read and output the Lux values:

```
import time
from Adafruit_TSL2561 import TSL2561

# Initialize the sensor
tsl = TSL2561()

while True:
    lux = tsl.calculate_lux()
    print("Current Lux: ", lux)
    time.sleep(1) # Delay for 1 second
```

Libraries on the internet make things so much easier these days. Back in the day, you had to study the datasheet for any new sensor you bought, and roll your own functions or assembly subroutines.

Improved Frame Processing: By integrating `cv2.GaussianBlur()` before object detection, the impact of noise and grain in low-light video frames was reduced. I found this especially useful when testing the Raspberry Pi Camera Module 3 NoIR (**Figure 3**, right) for the purpose of night surveillance. This model of the camera has no infrared filter, so it can see infrared illumination quite well. However, it still depends on the distance from your infrared illumination to the subject, and sometimes objects at a distance pose a challenge.

Calling this function gave the model significantly less noise to work with, even if it affected the sharpness of the input image. However, when you are detecting macro-scale objects and not trying to read license plates, the model doesn't care if your "human" is fuzzy — it will be up to the user to check the camera stream for detail once an alert is sounded. The function call is a single line of code:

```
frame = cv2.GaussianBlur(frame, (5, 5), 0)
```

Multi-Model Support: At first, I opted to choose a single model after trying several of them, but it was a bit frustrating that there wasn't one that was good for all setups and scenarios. Then I realized, as the gears turned in my head way too slowly, that you could actually just switch between models on the fly.

Incorporating a secondary detection model specialized for specific targets enhances precision. For instance, integrating a streamlined

YOLO model allows for more robust vehicle and pedestrian detection. Switching between models based on the scene context is managed as in this rather coarse example:

```
if scene == 'urban':
    model.load('yolo_city.tflite')
else:
    model.load('tensorflow_lite_default.tflite')
```

Reducing Latency: Unfortunately, as the complexity of detection increased, so did the latency. I tried implementing asynchronous model processing, which helped speed things up in terms of raw numbers, even if it wasn't very noticeable to me. Nonetheless, any help is welcome, so here's how to try that out:

```
from concurrent.futures import ThreadPoolExecutor

with ThreadPoolExecutor() as executor:
    future = executor.submit(process_frame, frame)
    result = future.result()
```

Here we see the use of the `ThreadPoolExecutor()` from the `concurrent.futures` module, which is part of Python's standard library. This approach is used to handle asynchronous execution of tasks in separate threads, which can be particularly useful for applications such as real-time video processing where responsiveness is crucial.

After the first line, which does the import, the next three do the following, respectively:

1. Create a `ThreadPoolExecutor` instance, which manages a pool of threads for executing calls asynchronously. The `with` statement ensures that the executor's resources are properly managed, automatically shutting down the executor when it is no longer needed.
2. The `submit()` method schedules the `process_frame()` function to be executed with `frame` as its argument. The function is run in a separate thread managed by the executor. `submit()` returns a `future` object, which represents the eventual result of the function call.
3. Blocks the main thread until the `process_frame()` function completes

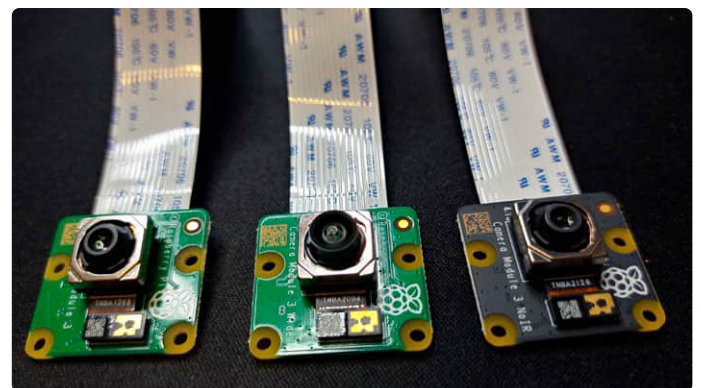


Figure 3: Besides the Wide and the regular Camera Module 3, I tried the NoIR variant for its ability to benefit from infrared illumination at night.

and returns a result. The `result()` method retrieves the outcome of the function when it finishes running. If the function raises an exception, it will be re-raised here.

By offloading intensive tasks to separate threads, the main program thread can remain responsive. This is especially important in GUI or real-time applications, where a sluggish response would negatively affect the user experience.

Deploying this system as part of a home surveillance network demonstrated its ability to differentiate between a person and a dog or a tree, but it's not as sophisticated as to employ face detection to differentiate between different people.


Where to Now?

Aside from all the deinterlacing and preprocessing I'll have to do to move forward on my pet old videos project, there are many other avenues left to explore using TensorFlow Lite, and I'm not sure which will be the most interesting, so I invite you to get in touch and let me know if there's an avenue you'd like to explore!

Some things we can try:

- **Image and video processing:** Still the one I'm busy on, it supports advanced image and video processing for applications such as image segmentation, where the model identifies different objects in the image and separates them from the background, or style transfer, which applies the style of one image to the content of another.
- **Real-time object detection and classification:** TensorFlow Lite can run models that detect and classify objects in real-time, which is useful for applications such as security cameras, automated quality control, and wildlife monitoring. Pre-trained models such as MobileNet can be adapted and deployed on devices to identify objects with high efficiency.
- **Speech recognition and audio processing:** TensorFlow Lite can handle models designed for speech recognition, enabling voice-activated applications, speech-to-text conversion, and other audio processing tasks. For me, this means rescuing forgotten history from the dusty media archives, but it also allows development of hands-free controls or aids for individuals with disabilities.
- **Natural language processing (NLP):** It supports lightweight NLP models that can perform tasks such as sentiment analysis, language detection, or even powering simple chatbots. This capability is particularly useful in applications requiring user interaction and feedback processing.
- **Gesture recognition:** By using TensorFlow Lite, you can develop systems that understand and interpret human gestures as commands, enabling interactive installations or enhancing user interfaces in devices.

- **Anomaly detection:** TensorFlow Lite can be used for anomaly or outlier detection in time-series data, which is valuable for predictive maintenance in industrial settings or monitoring systems like detecting irregular heartbeats in health devices.
- **Pose estimation:** The framework is capable of running pose estimation models that detect human figures and their postures in images or video. This can be applied in sports analytics, fitness apps, and advanced interactive applications, as we were able to see happening in real-time during our recent visit to embedded world 2024.

With the enhancements to TensorFlow Lite on Raspberry Pi, we're making solid progress in object and text detection, if not with low-resolution and interlaced video. These improvements could help hobbyists effectively manage real-time applications and are particularly useful for projects like home surveillance and media archiving. There's plenty more to explore and refine, and I look forward to seeing how we can further adapt these tools to our needs. 

230181-G-01

Questions or Comments?

Do you have technical questions or comments about this article? Email the author at brian.williams@elektor.com.



About the Author

Brian Tristam Williams has been fascinated with computers and electronics since he got his first "microcomputer" at age 10. His journey with Elektor Magazine began when he bought his first issue at 16, and since then, he's been following the world of electronics and computers, constantly exploring and learning. He started working at Elektor in 2010, and nowadays, he's keen on keeping up with the newest trends in tech, particularly focusing on artificial intelligence and single-board computers such as Raspberry Pi.

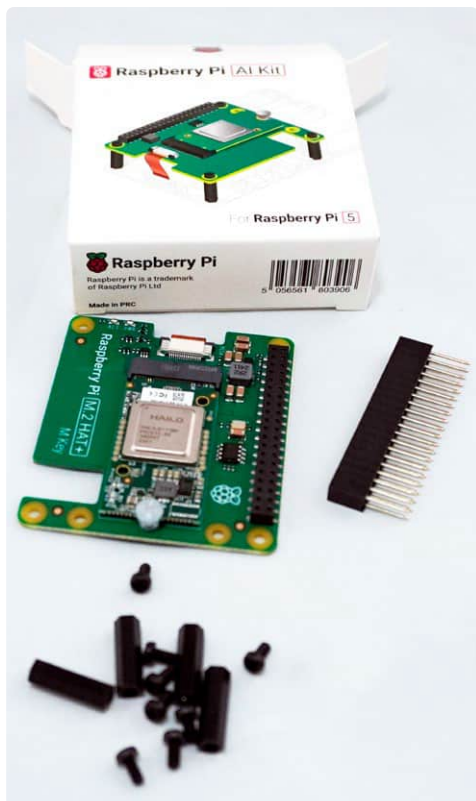


Related Products

- **Raspberry Pi 5 (4 GB)**
www.elektor.com/20598
- **Raspberry Pi Camera Module 3**
www.elektor.com/20362
- **Raspberry Pi Camera Module 3 NoIR**
www.elektor.com/20363

WEB LINK

[1] Adafruit TSL2561 library for Python on the Raspberry Pi: <https://github.com/adafruit/TSL2561-Arduino-Library>



▲ Figure 1: Contents of the Raspberry Pi AI Kit box.

Raspberry Pi has rolled out a kit consisting of its M.2 HAT+ bundled with the Hailo-8L AI accelerator. This new combo brings serious AI horsepower to the Raspberry Pi 5, giving developers and engineers some impressive inferencing performance to play with. Let's take a closer look at the Raspberry Pi AI Kit.

Why Live on the Edge?

Edge computing processes data locally on devices rather than relying on cloud-based servers. This approach significantly reduces latency, ensuring faster response times and improving reliability. It's crucial for applications requiring real-time decision-making, such as robotics, autonomous vehicles, and smart home devices. By managing data at the edge, one can enhance performance and reduce bandwidth expenses, latency, and enhance data security.

The catch is that being right on the edge typically involves hardware with nowhere near the power of those server farms up in the cloud. The Raspberry Pi AI kit aims to take on that problem, with the integration of the Hailo-8L with Raspberry Pi's M.2 HAT+ exemplifying this shift toward more efficient and responsive AI-driven solutions.

In a recent chat, Eben Upton, co-founder of Raspberry Pi, highlighted the growing importance of edge computing: "As more devices become intelligent, the demand for local AI processing increases. Our goal with this new AI kit is to make high-performance computing more accessible to developers working on the edge."

Raspberry Pi Goes AI

A Handy Kit Integrates AI Accelerator With M.2 HAT+

By Brian Tristam Williams (Elektor)

The Raspberry Pi AI Kit, featuring the M.2 HAT+ and the Hailo-8L AI accelerator board, delivers 13 TOPS of performance to the Raspberry Pi 5. This makes it perfect for real-time AI applications, edge computing, robotics, and computer vision. The integration process is straightforward, and there is extensive software support available.

What's in the Box?

In the package (**Figure 1**), you'll find a Raspberry Pi M.2 HAT+ with already-connected Hailo-8L M.2 accelerator board, 16 mm stacking header, four spacers and eight screws. The Hailo-8L comes already mounted to the Raspberry Pi M.2 HAT+ (**Figure 2**).



◀ Figure 2: The Kit consists of the Hailo-8L chip on its M.2-format accelerator board connected to the Raspberry Pi M.2 HAT+.

Figure 3: The accelerator board provides a way for the Hailo-8L chip to be connected to a PCIe interface via an M.2 connector.



The Hailo-8L Chip

So, what's the big deal with the Hailo-8L? For starters, this AI accelerator is a beast, making it perfect for real-time AI tasks, which means you can tackle projects in robotics, computer vision, smart home devices, and industrial automation without it breaking a sweat. The Hailo M.2 M-key 2242 footprint (22 mm × 42 mm) accelerator board (**Figure 3**) makes it super easy to tap into this power, leveraging the versatility of the Raspberry Pi 5. Plug it into the M.2 HAT+, which in turn connects to the Raspberry Pi 5's PCIe port via a 30 mm ribbon cable (**Figure 4**).

The Hailo-8L AI accelerator is designed to deliver data center-class performance to edge devices, boasting up to 13 TOPS (trillions of operations per second). This makes it an ideal fit for entry-level products that require AI capacity in the local area. What sets the Hailo-8L apart is its superior area and power efficiency, making it highly competitive compared to other solutions in its category.

It features low-latency, high-efficiency processing capable of managing complex pipelines with multiple real-time streams and concurrent processing of

various models and AI tasks. This accelerator is also compatible with the Hailo-8's comprehensive software suite, ensuring seamless upgrades to higher AI capacities in the future.

The Hailo-8L AI accelerator includes a comprehensive software suite with Hailo device drivers, *HailoRT*, and *HailoTappas*, which can be easily installed via the *apt* package manager. This ensures seamless setup and operation.

AI acceleration support is fully integrated with Raspberry Pi's camera software stack, including support for *libcamera*, *rpicas-apps*, and *picamera2*. This allows for advanced image processing and AI applications directly on the Raspberry Pi 5.

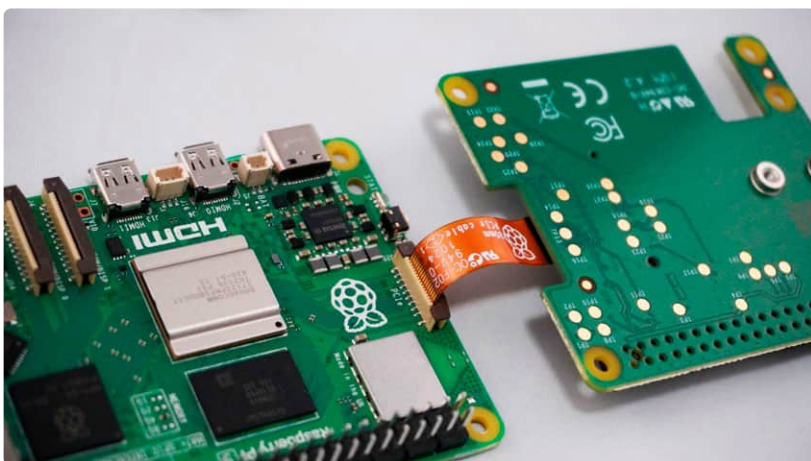
Key Features and Benefits:

- **High Efficiency:** The Hailo-8L is all about high-performance AI processing with minimal power draw. This is crucial for edge devices, where you need to balance performance and energy efficiency.
- **Real-Time AI Processing:** With its 13 TOPS, the Hailo-8L handles complex tasks such as object detection, image classification, and speech recognition in real-time. Your applications can be more responsive and smarter.
- **Seamless Integration:** The kit comes pre-assembled with the Hailo-8L. Just plug it into your Raspberry Pi 5, and you're ready to go. No fuss, no hassle.
- **Robust Software Support:** Fully integrated into the Raspberry Pi OS environment, you can easily install Hailo's software packages via the *apt* package manager. This includes device drivers and neural network libraries, so you can hit the ground running with your AI projects.

This setup pairs the affordability and versatility of Raspberry Pi with the advanced AI chops of the Hailo-8L. The result? You can run sophisticated AI models right at the edge, cutting down on latency and bandwidth compared to cloud-based solutions.

The Hailo-8L supports popular AI frameworks such as TensorFlow, TensorFlow Lite, Keras, PyTorch, and ONNX, and is compatible with ARM host architectures. This makes it ideal for deploying sophisticated AI models on the Raspberry Pi 5, and opens up a whole new area of possibilities for what we can achieve with our Raspberry Pi setups.

Figure 4: The Raspberry Pi AI Kit connects to the Raspberry Pi 5's PCIe connector.





As more devices become intelligent, the demand for local AI processing increases. Our goal with this new AI kit is to make high-performance computing more accessible to developers working on the edge.

Eben Upton

Upton explained the decision to collaborate with Hailo: “What we found exciting about Hailo was its high performance and efficiency. The Hailo-8L brings 13 TOPS to the table, significantly more than previous solutions. This, combined with its excellent tooling for model conversion, made it a natural fit for our edge AI ambitions.”

Development and Challenges

Asked if he foresaw the M.2 HAT being used for AI accelerators besides NVMe SSDs, he said, “When we launched the Raspberry Pi 5, we initially thought most uses for the M.2 HAT would be for storage. However, we quickly realized there was significant interest in AI accelerators, network cards, and graphics solutions.”

As for the inevitable obstacles that are encountered in putting together a sophisticated pairing such as this, he recalled, “One challenge we faced nearing launch was optimizing the thermals. Initially, we didn’t have a thermal pad between the accelerator and the baseboard, but we included it in the final version to improve heat dissipation.” Despite these hurdles, the collaboration between Raspberry Pi and Hailo proved fruitful, leveraging each company’s strengths to deliver a robust and efficient product.

Upton also mentioned the difficulty of applying AI acceleration due to the rapidly changing architectures of models: “One of the challenges I think we have with GenAI at the moment is that it’s actually very hard to apply acceleration. Things need to settle down before you can design an accelerator that is meaningfully more capable than a CPU or GPU without sacrificing flexibility.”

Use Cases and Ideal Applications

Upton shared his excitement about the future of AI on the Raspberry Pi platform: “We’re going to see some amazing applications in industrial robotics and smart home devices. The potential is enormous, and this AI kit will help unlock that for our users,” Upton said. He also highlighted the popularity of cameras as accessories for the Raspberry Pi: “Cameras are an incredibly popular accessory for Raspberry Pi, and being able to apply more intelligence to those camera applications

is a big deal. This AI accelerator is perfect for tasks like object detection and image classification.”

The Hailo-8L is optimized for vision-related tasks, making it ideal for applications such as surveillance, automated quality inspection in manufacturing, and healthcare imaging. “One of the challenges with GenAI at the moment is the rapidly changing architectures of models. However, the Hailo-8L is designed to handle more stable, vision-related tasks efficiently.”

The Raspberry Pi and Hailo Partnership

Raspberry Pi is all about making high-performance computing accessible and affordable. Hailo, on the other hand, specializes in AI processors that bring data center-class performance to edge devices. This partnership highlights what’s possible when you combine cutting-edge hardware innovation with AI acceleration.

Upton also reflected on the educational impact of their products: “Our aim has always been to put programmable hardware in kids’ hands and see what happens. It’s about nurturing curiosity and enabling the next generation of engineers and developers.”

For those of us always looking to push the boundaries with AI at the edge, the new Raspberry Pi AI Kit is a game-changer. It opens up a whole new realm of possibilities for what we can achieve with our Raspberry Pi setups. ◀

240298-01

Questions or Comments?

Do you have any questions or comments about the Raspberry Pi AI Kit? Contact the author at brian.williams@elektor.com.



Related Products

▶ **Raspberry Pi AI Kit**
www.elektor.com/20879



Weather Station Sensors

Which One Should You Choose?

By Saad Imtiaz (Elektor)

When building a weather station, you have so many options of sensors to choose from. It should be accurate, reliable and not too costly, and also easy to control by the microcontroller of your choice. In this article, we will provide an in-depth comparison of some environmental sensors.

The selection of sensors is a critical step in the process of assembling a personal weather station, and it greatly influences the accuracy and reliability of the data collected. Given the numerous options available, it is imperative to comprehend the strengths and limitations of various environmental sensors. This article offers a detailed comparison of various sensors that can be used in weather stations, focusing on their performance, integration ease, and overall reliability. By explaining the technical aspects of these sensors, the aim is to assist in making informed decisions that enhance the precision of environmental readings. In **Figure 1** you can see a fully equipped weather station featured in Elektor [1].

Selection Criteria for Sensor Comparison

Before starting the comparison of environmental sensors for weather stations, it is necessary to establish precise guidelines to navigate the wide-ranging array of sensors available. The focus will be narrowed using the following defined criteria:

Grade: The analysis will be limited to sensors designed for consumer applications. These sensors strike a balance between performance and price, making them ideal for hobbyists and enthusiasts looking to build or enhance personal weather stations.

Availability: Only sensors that are readily available on the market will be considered. Accessibility is important so that people can use the information they get right away, without having to wait for it to be available or buy it from someone else.

Module-Based Solutions: The comparison will focus on sensors that are provided as modules. This approach facilitates ease of integration, as modules typically include necessary signal conditioning and interface circuitry, making them suitable for a wide



Figure 1: Weather Station project by Elektor's former engineer Mathias Claussen [1].

array of applications without the need for complex electronics design.

Price Range: The scope of the comparison will be sensors with a retail price between €2 and €20. This price bracket is chosen to cater to a broad audience, including hobbyists and for educational purposes, ensuring that the deployment of a personal weather station remains an affordable project.

Interoperability and Integration Ease: An essential aspect of our guidelines is the ease of integration into existing systems. Sensors that offer straightforward compatibility with popular microcontrollers and development platforms, such as Arduino and Raspberry Pi, will be prioritized. This ensures that users can easily incorporate these sensors into their projects with minimal additional learning curve or system reconfiguration.

Sensors

In a typical weather station, especially those designed for personal or educational use, a core set of environmental sensors is commonly employed to monitor various atmospheric parameters. The focus will be on the following types of sensors, each playing a key role in the functionality of a weather station:

Temperature and Humidity Sensors: These sensors are fundamental for measuring the air's warmth and moisture levels.

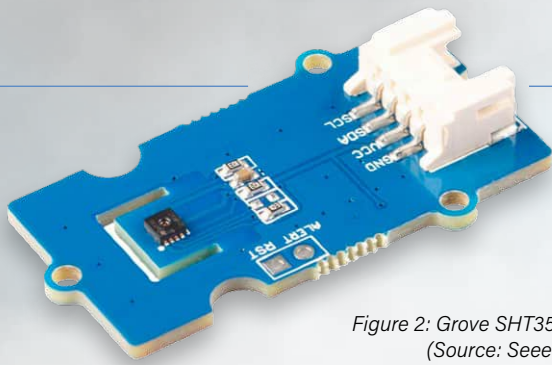


Figure 2: Grove SHT35 Module.
(Source: Seeed Studio)

Temperature is a primary weather parameter, while humidity readings are crucial for understanding precipitation potential and comfort levels.

Barometric Pressure Sensors: Pressure measurements are key to predicting weather changes. A barometric pressure sensor offers insights into weather trends, such as approaching fronts or storms, by detecting variations in atmospheric pressure.

Rainfall Sensors: To quantify precipitation, rainfall sensors are employed. They typically measure the amount of rain over a specific period, providing data essential for understanding precipitation patterns and water cycle dynamics.

Wind Speed and Direction Sensors (Anemometers and Wind Vanes): These sensors measure wind's speed and direction, critical data for weather forecasting and studying wind patterns in a given location.

Solar Radiation Sensors: For a comprehensive weather analysis, measuring the sun's energy received at the Earth's surface is vital. Solar radiation sensors, or pyranometers, assess sunlight intensity, contributing to data on weather conditions and solar power potential.

UV Index Sensors: In some advanced or specialized weather stations, UV index sensors monitor the sun's ultraviolet radiation level. This information is crucial for health and environmental monitoring, indicating the risk of sunburn and the effect of UV radiation on ecosystems.

These sensors form the backbone of a weather station, providing a comprehensive view of atmospheric conditions. This focus ensures the sensors are both affordable and easy to integrate for personal weather station projects.

Temperature and Humidity

This section compares several temperature and humidity sensors, focusing on their accuracy, operational ranges, and power consumption to aid in selecting the most suitable sensor for reliable and efficient environmental monitoring.

SHT35 and **SHT40** by Sensirion [2] are known for their high accuracy, with temperature readings accurate to 0.1°C and humidity readings accurate to 1.5% RH. In **Figure 2** the SHT35 Module by Seeed Studio is shown. Applications that need precise environmental measurements will benefit from this. However, this precision comes at a higher price point than the other options.

The SHT40 is a practical choice for projects where cost constraints are significant, as it offers a more cost-effective solution with slightly lower accuracy.

AHT20 sensor from Aosong Electronic [3] operates with a supply voltage of 2.2 to 5.5 V and measures humidity from 0 to 100% RH and temperature from -40 to +85°C. It offers accuracies of $\pm 2\%$ RH and $\pm 0.3^\circ\text{C}$, respectively. The sensor is very accurate and can respond in 5 s for temperature and 8 s for humidity. It uses the I²C serial protocol for communication with microcontrollers, making it compatible with many microcontrollers.

The **SHTC3** by Sensirion [4] sensor can measure temperature and humidity with an accuracy of 0.2°C and 2% RH. It operates within a temperature range of -40 to 125°C and a humidity range of 0 to 100% relative humidity, with response times of about 5 s. Operating on a voltage range of 1.62 to 3.6 V, it draws an ultra-low average current of 0.5 μA . This sensor has compact dimensions and I²C interface, making it suitable for straightforward integration. In **Figure 3** an SHTC3 Module by Soldered Electronics is shown.

Honeywell's **HIH6130** [5] offers a good balance of accuracy, $\pm 0.5^\circ\text{C}$ for temperature and $\pm 3\%$ RH for humidity with robustness, with a digital output. Its power consumption is moderate, typically around 450 μA during operation.

The **MPL3115A2** by NXP Semiconductors [6] is a precision altimeter sensor designed to measure pressure and temperature, enabling accurate altitude calculations. It operates within a pressure range of 20 to 110 kPa and a temperature range of -40 to 85°C, with a pressure accuracy of ± 0.4 kPa and temperature accuracy of $\pm 1.0^\circ\text{C}$. The sensor features an altitude resolution of 30 cm and a temperature resolution of 0.1°C, supporting both I²C and SPI communication interfaces. It has low-power consumption, typically just 40 μA in standard mode.

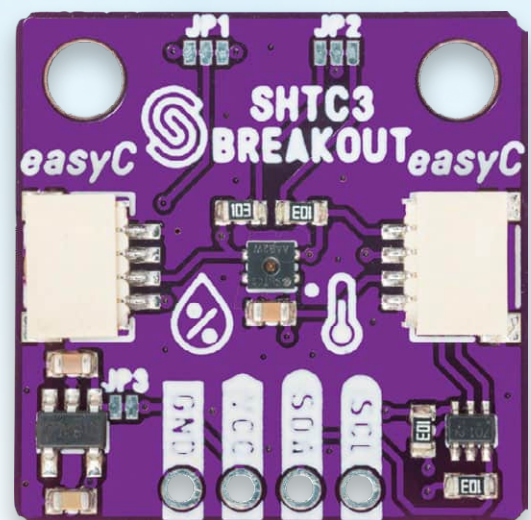


Figure 3: SHTC3 Temperature and Humidity Module.
(Source: Reichelt Elektronik)

Rain Fall, Wind Speed, and Direction Sensors

It is necessary to measure the weight of the rain in a container of a known size to get the correct amount of rain that happened. For doing this, there are a bunch of options available and also DIY solutions can be done to save costs.

For a DIY solution, one can 3D-print a container with an attachment with a load sensor. The load sensor will measure the weight of the water, and by calculating the weight of the tank the volume can be measured. Eventually, amounts of rain in ml can be recorded.

Various solutions for measuring the wind speed and direction, It would be recommended that to get a proper kit for it, which is commercially available as shown in the figure. But if one wants to make a DIY solution for wind speed measurement, it's also possible. For that, a 3D Print is required for the 3 cups that spin in the wind, and a vane that aligns with the direction of the wind. To measure the velocity of the wind, a magnetic encoder would be suitable. The magnet would be attached at the end of the 3 cups shaft, and the magnetic encoder would be attached on the adjacent side to measure the rpm of the shaft, thereby calculating the velocity of the wind. For finding the direction of the wind, a digital compass sensor can be used, which will easier to implement than an equivalent resistive method implemented in most of the commercial options available.



Comparative Analysis

In evaluating specific standout sensors based on key performance indicators, optimal choices tailored to distinct application needs can be identified:

Accuracy: The Sensirion SHT35 is the most accurate sensor for both temperature and humidity measurements, ideal for critical applications where precision is crucial.

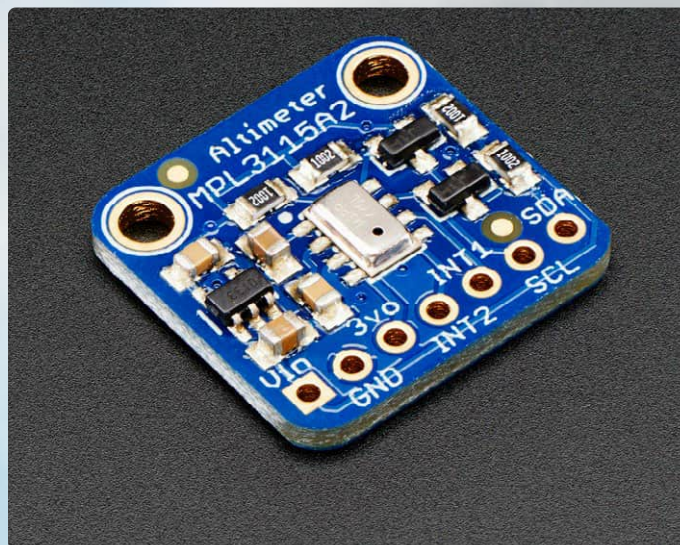


Figure 4: Adafruit MPL3115A2 Sensor Module. (Source: Adafruit)

Cost-Effectiveness: The Sensirion SHT40 offers great value, providing adequate accuracy at a reduced cost. This makes it particularly attractive for budget projects. The **MPL3115A2** by NXP Semiconductors is also a suitable choice because it has a reasonable price and good accuracy. In **Figure 4**, the MPL3115A2 Module by Adafruit is shown.

Power Consumption: For energy efficiency, the Sensirion SHTC3 leads the pack. Consuming only 0.5 μ A, it stands out in applications that require minimal power draw, such as battery-operated devices or remote sensing units, making it a good choice for power-sensitive designs.

Each sensor has its merits and should be chosen based on specific project requirements, including the environmental conditions it will operate in and the degree of precision needed. This choice should also consider the integration ease with existing systems and the overall cost implications of the sensor and associated peripherals.

For more detailed comparisons, it would be essential to review each sensor's datasheet. This would provide precise information on their electrical characteristics, environmental durability, size, power requirements, and more, allowing for an informed decision based on the specific needs of the project. An overall general comparison of these sensors is shown in **Table 1**.

Multifunction Sensors

Multi-parameter sensors like the BME280 and BME688 by Bosch Sensortec series are advantageous for several reasons. Firstly, they offer a streamlined solution for environmental monitoring, measuring variables such as temperature, humidity, pressure, and gas levels in a single compact device.

Table 1: Comparison of temperature and humidity sensors.

Sensor Name	Manufacturer	Measures	Accuracy (Temp/Hum/Pressure)	Operating Range (Temp/Hum/Pressure)	Current Consumption (Average)	Additional Features
SHT35	Sensirion	Temperature, Humidity	±0.1°C / ±1.5% RH	-40 to 125 °C / 0 to 100% RH	1.5 µA	High accuracy, Factory calibration
DHT22	Aosong Electronics	Temperature, Humidity	±0.5°C / ±2.5% RH	-40 to 80 °C / 0 to 100% RH	~	Budget friendly
HTU21D	TE Connectivity	Temperature, Humidity	±0.3°C / ±2% RH	-40 to 125 °C / 0 to 100% RH	2.7 µA	Fast response, I ² C interface
DS18B20	Maxim Integrated	Temperature	±0.5°C	-55 to 125°C / NA	1 µA	Waterproof models available
MPL3115A2	NXP Semiconductors	Temperature, Pressure	±0.3°C / ±0.4 kPa	-40 to 85°C / 20 to 110 kPa	2 µA	Altimeter functionality
LPS22HB	STMicroelectronics	Pressure	±0.1 hPa	NA / 260 to 1260 hPa	3 µA	Ultra-compact, high-resolution sensor
MS5611	TE Connectivity	Pressure	±1.5 hPa	NA / 10 to 1200 hPa	1.2 µA	High resolution, low power consumption
SHTC3	Sensirion	Temperature, Humidity	±0.2°C / ±2% RH	-40 to 125 °C / 0 to 100% RH	0.5 µA	I ² C interface, 5 s response time

Bosch Sensortec **BME280** [7] is a widely used sensor for temperature, humidity, and pressure measurement. It achieves temperature accuracy within ±1.0°C, humidity accuracy within ±3%, and pressure accuracy within ±1.0 hPa. Operating across a temperature range of -40 to 85°C, humidity from 0 to 100% RH, and pressure from 300 to 1100 hPa, it also features a low power consumption, drawing as little as 0.1 µA in sleep mode. The sensor is ideal for portable weather stations and home automation systems due to its precision and low energy demand.

Bosch Sensortec **BME688** [8] builds on the capabilities of the BME280 by adding volatile organic compounds (VOCs) detection, making it suitable for indoor air quality monitoring. This sensor maintains the same temperature and humidity accuracy as the BME280 but offers enhanced pressure accuracy of ±0.6 hPa. The BME688 also includes gas detection and also AI capabilities, which requires a bit more power but still maintains efficiency suitable for battery-operated devices. The BME688 can be perfect for indoor environment, sensor nodes and network applications.

Honeywell **HPM Series** [9] focuses on particulate matter detection, measuring PM2.5 and PM10 with accuracies of ±5 µg/m³ and ±10 µg/m³, respectively. Designed for environmental health monitoring, it operates within a range of 0 to 1,000 µg/m³. This sensor utilizes a fan to draw in air for sampling, leading to a higher current draw of about 80 mA during operation, which is higher compared to the Bosch sensors but justified by its specific application in air quality assessment.

Sensirion **SGP40** [10] is a compact sensor that measures indoor air quality by detecting total volatile organic compounds (TVOCs) and equivalent CO₂ levels. It offers a typical accuracy of 15 ppb

for TVOC measurement. The SGP40's average current consumption is around 2.6 mA, with peak currents up to 3 mA at 3.3 V when measuring.

Bosch Sensortec **BME680** [11] integrates measurements for temperature, humidity, pressure, and volatile organic compounds (VOCs) into a single device. With accuracies of ±1.0°C for temperature, ±3% RH for humidity, and ±0.12 hPa for pressure, it excels in precision. The sensor also features a gas sensor for VOC detection, enhancing its functionality for detailed environmental monitoring. Operating within a range of -40 to 85°C for temperature, 0 to 100% RH for humidity, and 300 to 1100 hPa for pressure, the BME680 is energy-efficient, consuming less than 0.1 mA in standard mode. In **Figure 5** a module of BME 680 by Joy-IT is shown.

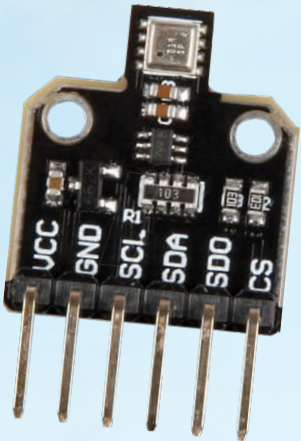


Figure 5: Bosch BME680 Sensor Module.
(Source: Reichelt Elektronik)

These integrated sensors simplify design and deployment, are space-efficient, and reduce the complexity of circuitry. Moreover, they are typically more power-efficient and cost-effective than using separate sensors for each environmental variable. These sensors with unified interface, such as Grove connectors, further enhances ease of use, enabling quick integration into projects without extensive wiring or technical setup. This integration is particularly beneficial for applications where space and power are at a premium, such as wearable technology, portable devices, and compact environmental monitoring stations. Comparison of these sensors is shown in **Table 2**.

Table 2: Comparison of multifunction sensors.

Sensor Name	Manufacturer	Measures	Accuracy (Temp/Hum/Pressure)	Operating Range (Temp/Hum/Pressure)	Current Consumption (Average)	Additional Features
BME280	Bosch Sensortec	Temp., Humidity, Pressure	Temp. $\pm 1.0\text{ }^{\circ}\text{C}$, Hum. $\pm 3\%$, Press. $\pm 1.0\text{ hPa}$	Temp. $-40\text{ to }85\text{ }^{\circ}\text{C}$, Hum. $0\text{ to }100\%\text{ RH}$, Press. $300\text{ to }1100\text{ hPa}$	$<0.1\text{ }\mu\text{A}$	Low power, portable stations, home automation
BME688	Bosch Sensortec	Temp., Humidity, Pressure, VOCs	Temp. $\pm 1.0\text{ }^{\circ}\text{C}$, Hum. $\pm 3\%$, Press. $\pm 0.6\text{ hPa}$	Similar to BME280, includes gas detection	$3.7\text{ }\mu\text{A}$	Indoor air quality, VOC detection, AI capabilities
HPM Series	Honeywell	Particulate Matter (PM2.5, PM10)	PM2.5 $\pm 5\text{ }\mu\text{g}/\text{m}^3$, PM10 $\pm 10\text{ }\mu\text{g}/\text{m}^3$	$0\text{ to }1,000\text{ }\mu\text{g}/\text{m}^3$	$\sim 80\text{ mA}$	Environmental health monitoring
SGP40	Sensirion	TVOCs, CO2 Levels (equivalent)	TVOC $\pm 15\text{ ppb}$	Temp. $-10\text{ to }50\text{ }^{\circ}\text{C}$, Hum. $0\text{ to }90\%\text{ RH}$	2.6 mA	Indoor air quality, compact design
BME680	Bosch Sensortec	Temp., Humidity, Pressure, VOCs	Temp. $\pm 1.0\text{ }^{\circ}\text{C}$, Hum. $\pm 3\%$, Press. $\pm 0.12\text{ hPa}$	Temp. $-40\text{ to }85\text{ }^{\circ}\text{C}$, Hum. $0\text{ to }100\%\text{ RH}$, Press. $300\text{ to }1100\text{ hPa}$	$<0.1\text{ mA}$	Detailed environmental monitoring, energy-efficient

Final Remarks

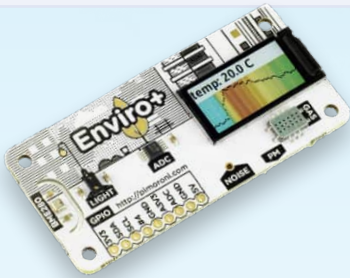
The selection of sensors for an independent weather station requires careful evaluation of technical parameters, integration ease, and cost-effectiveness. This article has conducted a thorough evaluation of numerous sensors, focusing on those that are accessible, inexpensive, and capable of providing reliable information. As technology advances, the accuracy, and utility of these sensors are likely to improve, enhancing opportunities for weather monitoring. For accurate and current information, refer to the latest datasheets and manufacturer releases. This strategy ensures the weather station remains a reliable tool for environmental monitoring. ◀

240002-01



Related Products

- ▶ Wind and Rain Sensors for Weather Station
www.elektor.com/20234
- ▶ Enviro+ Environmental Monitoring Station
www.elektor.com/18975
- ▶ SparkFun Weather Shield
www.elektor.com/19089



Questions or Comments?

If you have questions about this article, feel free to email the author at saad.imtiaz@elektor.com or the Elektor editorial team at editor@elektor.com.

WEB LINKS

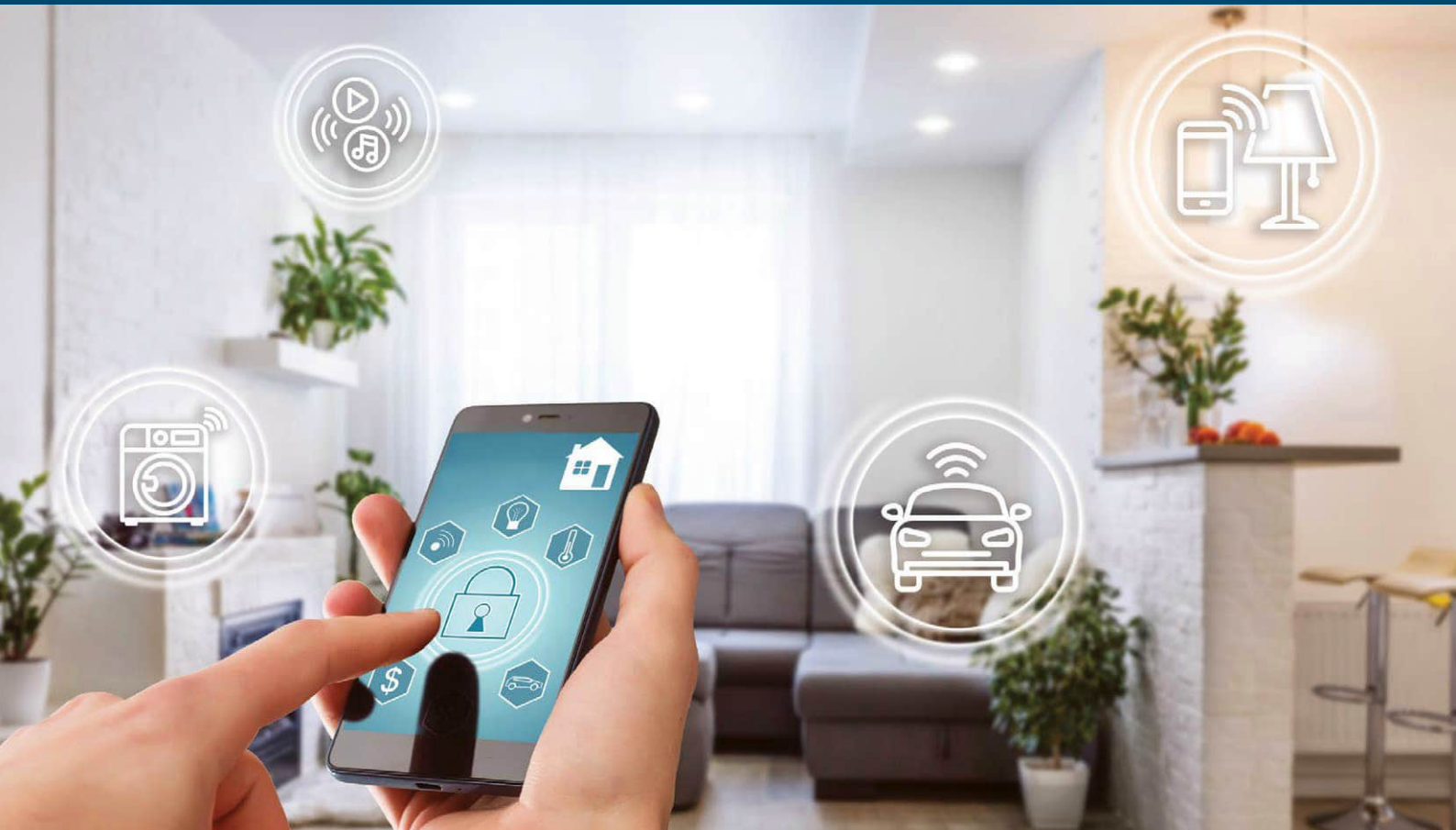
- [1] Mathias Claussen, "Open-Network Weather Station Mk.2," Elektor 5-6/2020:
<https://www.elektormagazine.com/magazine/elektor-146/58542>
- [2] Sensirion SHT3X-DIS-F | Datasheet: <https://sensirion.com/resource/datasheet/sht3x-d>
- [3] Aosong Electronic AHT20 | Datasheet: <http://www.aosong.com/en/products-32.html>
- [4] Sensirion SHTC3 | Datasheet: <https://sensirion.com/resource/datasheet/shtc3>
- [5] Honeywell HIH6130 | Datasheet: https://eu.mouser.com/datasheet/2/187/HWSC_S_A0012940693_1-3073215.pdf
- [6] NXP MPL3115A2S | Datasheet: <https://www.nxp.com/docs/en/data-sheet/MPL3115A2S.pdf>
- [7] Bosch BME 280 | Datasheet <https://tinyurl.com/BME-280-Datasheet>
- [8] Bosch BME 688 | Datasheet <https://tinyurl.com/BME-688-Datasheet>
- [9] Honeywell HPM Series <https://tinyurl.com/hpm-series>
- [10] Sensirion SGP40 | Datasheet <https://sensirion.com/resource/datasheet/sgp40>
- [11] Bosch BME680 | Datasheet <https://tinyurl.com/BME-680-Datasheet>

Microchip is...

IoT

Sensor Interface <
Microcontrollers <
Microprocessors <

Security <
Connectivity <
Cloud Services <



- Automotive
- Home Appliance
- Lighting
- Medical
- Smart Energy/Metering
- Wireless Audio



microchip.com/loT



The Microchip name and logo and the Microchip logo are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries. All other trademarks are the property of their registered owners.
© 2023 Microchip Technology Inc. All rights reserved.
MEC2481A-UK-02-23

AI-Based Water Meter Reading (1)

Get Your Old Meter Onto the IoT!

By Daniel Scaini (Italy)

Smart water meters have already been on the market for some time, but replacing our old meters is often not so easy, for technical or bureaucratic reasons. With this project, an analog meter of any kind can be turned into a digital one using an ESP32-CAM platform and an artificial intelligence (AI) system. Because we will also unveil a lot of background in this article, we have split it in two parts.

Italy is one of the European countries that withdraws and consumes the most water for civilian use, second only to Greece. These numbers are worrying, especially considering the water crises that punctually hit our country. Average availability, in fact, has declined by about 19% in the last three years, worsened by increasingly hot temperatures and a shortage of rainfall. This problem is not unique to Italy, and several nations are taking drastic countermeasures, such as limiting consumption or allocating funds to renovate private or state-owned distribution facilities in order to avoid leaks.

In addition, given the continuous evolution of technology, several distributors have been taking steps for a few years now to make the reading of civilian meters more efficient as well. This measure serves to avoid wastage of the raw material, but also to reduce the reading operations performed house by house by operators.

In this article, we are going to address this issue: reading an analog meter to make it digital and have it transmit the values to us or to one of our servers. The whole thing is based on an ESP32-CAM, a platform already known to many, and an artificial intelligence (AI) system that can go and detect and translate photos into potential readings.

Architecture

AI systems have entered our daily lives in a big way — just think of voice assistants or image recognition. For the complex computations that an AI system has to deal with, it is possible to rely on cloud computing on dedicated online platforms, or to perform them directly on the chip, in so-called edge computing. With the increasing enhancement of processors, this second mode of processing is expanding and is the basis of our project.

In this, an AI network and an ESP32-CAM will cooperate in order to be able to give

Elettronica In
WWW.ELETTRONICA.IT

the user a digital result obtained by digitally photographing a classic analog water meter. The recognition and digitization is done by the ESP32-CAM using a convolution neural network (CNN), which we will deal with later in the article.

The first step in implementing our project is to install the firmware on our ESP32-CAM. Next, we will need a calibration phase in which we will identify the areas designated for recognition of the numbers and indicators on our counter. Once this is completed, we will have all the information available to send it digitally where we wish.

Neural Networks

Neural networks are computational models inspired by the workings of the human brain. They are artificial intelligence systems that attempt to emulate the way the brain processes information. Neural networks are used in machine learning, which is a field of study concerned with training computers to perform certain tasks without being explicitly programmed for that purpose.

Neural networks play a critical role in deep learning models, which is a branch of machine learning that focuses on processing complex data. They are composed of computational

units called *artificial neurons* or *nodes*. These neurons are connected to each other through artificial connections called *weights*. Each connection has an associated numerical value, which represents the importance of the connection to the model.

Neural networks are organized in layers, with one or more layers hidden between the input layer and the output layer. The first layer, called the *input layer*, receives the input data. The middle layers, called *hidden layers*, process the information through their neurons. Finally, the last layer, called the *output layer*, produces the desired results, as schematized in **Figure 1**.

Neural networks can be used in a wide range of applications, such as machine translation, natural language processing, medical diagnosis and, in our case, image recognition. In this specific case, during the training process, filters are applied to the image at different resolutions, and the output of each processed image is used as input for the next layer. The filters start with basic features, such as brightness or edges, and become increasingly complex as they progress, including features that uniquely define the object.

CNNs are a specific type of neural network designed for efficient processing of structured data, such as images or video. What distinguishes CNNs from traditional neural networks is the use of an operation called *convolution*. During convolution, a small window called a *filter* or *kernel* slides over the input image and a series of mathemat-

ical operations are applied to each portion of the image. This process allows CNN to automatically extract relevant features, such as edges, textures or patterns, from the images efficiently.

These networks also consist of an input layer, an output layer, and many intermediate layers called *hidden layers*. They contain three main types of layers, namely:

- > convolutional layer
- > pooling layer
- > fully-connected (FC) layer

The convolutional layer is the first layer of a convolutional neural network. It is responsible for extracting the salient features from the input image. Convolutional layers may be followed by other convolutional layers or pooling layers. Subsequent convolutional layers continue to process features extracted from previous layers, allowing the network to learn increasingly complex and abstract features.

The fully connected layer, also called the output layer, is the last layer of a CNN. At

this level, extracted features are used to make predictions or classifications. This level is responsible for providing the final output of the convolutional neural network.

At each level, the complexity of the CNN increases as the network learns increasingly sophisticated and abstract features of the input image. Moreover, as one proceeds through the convolutional levels, the portion of the image that is identified and analyzed in detail by the neural network increases, as illustrated in **Figure 2**. Unlike a traditional neural network, a CNN has shared weights and biases that are the same for all neurons hidden in a given layer. After learning the features in multiple layers, the architecture of a CNN moves on to classification.

The penultimate layer is a fully connected layer that generates a vector of size K (where K is the number of predictable classes) and contains the probabilities for each class of any classified image. The last layer of the CNN architecture uses a classification layer to provide the output of the final classification. Usually, you have pre-trained models of these networks, which weigh several megabytes

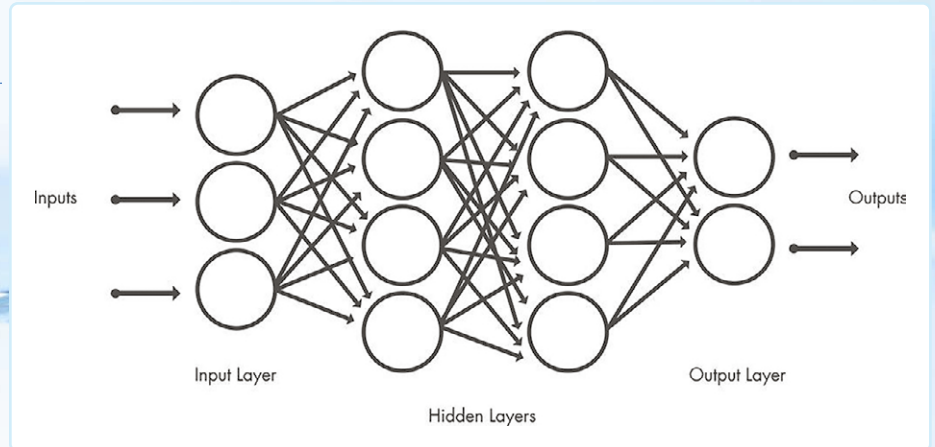


Figure 1: Neural networks are composed of an input layer, an output layer, and many intermediate layers called hidden layers. (Source for all the pictures in this article: Elettronica In — <https://futura.net.it>)

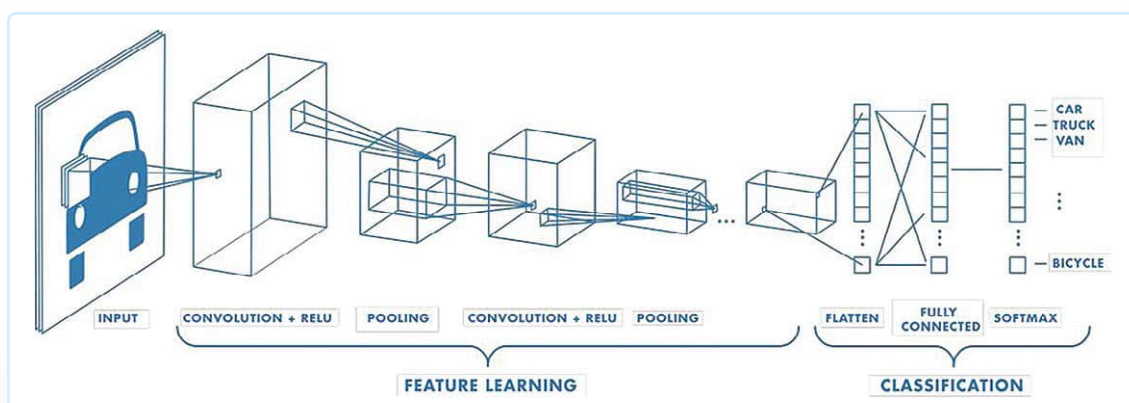


Figure 2: At each layer, the complexity of the CNN increases.

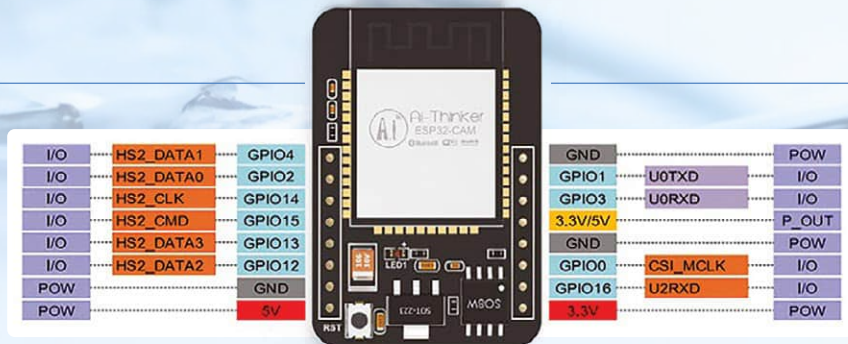


Figure 3: Pinout of the ESP32-CAM module.

or even gigabytes. In our case, this will be a couple of megabytes but will be saved on and SD card so as not to burden the chip's internal memory.

Specifically, there are several files present in the config folder with the extension *.tflite*, which stands for TensorFlow Lite. TensorFlow, born out of Google Brain in 2015, has become a reference library for creating Deep Learning models. Initially developed internally by Google, it was made open source and quickly gained popularity among the machine learning community. It offers a wide range of machine learning and deep learning models and algorithms, known as neural networks, and makes them available to developers through an intuitive API.

TensorFlow leverages the power of Python or JavaScript to provide an easy-to-use programming interface for creating applications. Meanwhile, the execution of such applications takes place in C++, which allows for high computational performance. This makes TensorFlow a versatile choice for large-scale machine learning projects.

Models trained with TensorFlow can also be implemented on mobile or edge computing devices, as in our case, as well as on iOS or Android operating systems. The TensorFlow ecosystem offers tools such as TensorFlow Lite, which optimizes TensorFlow models to run efficiently on such devices. With TensorFlow Lite, a tradeoff can be made between the size of the model and its accuracy. A smaller model may take up less space, such as 12 MB instead of 25 MB, or perhaps more than 100 MB, but it may have a slight reduction in accuracy. However, this loss in accuracy is often negligible, considering the speed and energy efficiency advantages that the compressed model offers.

In our case, this model trained with TensorFlow will be used to distinguish and recognize

numbers within the counter and to determine the direction of the indicators. With the help of TensorFlow, we will be able to leverage the power of deep learning for data analysis and artificial intelligence, improving the efficiency and accuracy of our applications.

Hardware

As mentioned, the chosen platform is AI-Thinker's ESP32-CAM, which, thanks to its compact size (40.5×27×4.5 mm) could be the Holy Grail for any maker. Successor to the well-known ESP8266, which we will discuss later, it consists of an ESP32 module equipped with a connector — whose pinout is visible in **Figure 3** — to accommodate a separate camera module and a slot for and SD card of up to 4 GB. In detail, we can see a programmable microcontroller with built-in Wi-Fi and Bluetooth, and additional external RAM of up to 4 MB.

Not only that, the new camera connector can embed either an OV2640 or OV7670 module; the former comes with the module itself and has a resolution of 2 megapixels. Its SPI speed is 8 MHz and the frame buffer size is 384 KB. Its normal power consumption is 70 mA, while in power-saving mode, it drains only 20 mA, making it desirable for its low-power feature as well.

There is also a high-brightness LED on board (**Figure 4**), which can be used as a flash or illuminator of the scene to be filmed. This feature is critical for our project, since it will be placed in environments that are normally very dark. We will see that it will be possible to modulate this light to achieve the desired result. Of course, should the previous one not suffice, the numerous GPIOs mounted on the board allow us to possibly mount an external illuminator in the case of low light.

For small applications, this chip comes across as very robust and with great processing power, relying on 2×32-bit cores



Figure 4: The ESP32-CAM is the heart of this project.

@ 120 MHz. This last feature allows it to have a fairly high frame-rate, obviously depending on the format and size: Indicatively we can achieve a throughput of up to 8 JPEGs in SVGA (800×600) per second.

Programming our ESP32-CAM module can be done in several ways: The most classic involves the use of an adapter of FTDI (**Figure 5**) which, thanks to the FT232RL interface, simulates RS232 and COM ports, thus allowing quick plug-and-play functionality.

Note from the editor: The module suggested in the "Related Products" frame at the end of this article — fully pin-compatible and available from the Elektor Store — embeds a Micro-USB connector with the related serial interface to the chipset, making any external serial adapters unnecessary.

In case you do not have the abovementioned adapter available, do not despair. In fact, you can program this module using any other module that has TX and RX pins. For example, from different models of the Arduino or ESP.

Wiring for Programming the ESP32-CAM

Just for the reason mentioned above, there can be different types of wiring, similar but different in some respects. In general, the ESP32-CAM module does not have any USB to interface directly with a PC, and that is precisely why we have to use an external module.

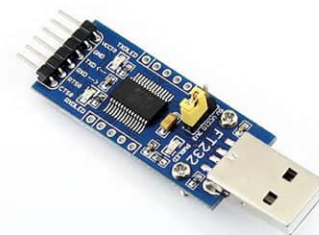


Figure 5: The USB-TTL converter with FT232RL.

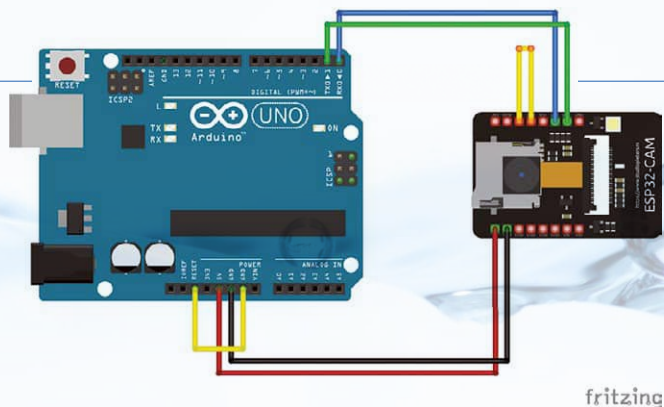


Figure 6: The ESP32-CAM module can be programmed with the Arduino UNO.

Table 1: Arduino UNO to ESP32-CAM Wiring.

Arduino ⇒ ESP32-CAM	Arduino	ESP32-CAM
TX ⇒ U0TXD	Reset ⇒ GND	GPIO0 ⇒ GND
RX ⇒ U0RXD		
5V ⇒ 5V		
GND ⇒ GND		

Let us start with the most classic of modules: the Arduino UNO. As a first step, we identify the transmit and receive PINs on our ESP32-CAM, called U0TXD and U0RXD, respectively. These will need to be connected directly to the TX and RX PINs on our Arduino. In the camera module, these pins also correspond to the GPIO1 and GPIO3 pins. Next, we short the ESP32-CAM's IO0 contact to the GND contact to ensure the proper communication mode. For the power supply, we connect the respective 5V and GND pins between the two platforms.

Finally, to reset the Arduino if needed, we short the RESET pin to the GND pin. Summarizing, we get the diagram in **Figure 6**, also shown in **Table 1**.

If we do not have an Arduino UNO but rather an ESP8266, the wiring changes, but very little. In fact, the only change worth mentioning is the contacts to be shorted on our programmer module: They are no longer RESET and GND but EN and GND. The wiring is shown in **Figure 7** and indicated in **Table 2** as well.

Finally, the simplest wiring of all. In fact, if we own a USB-TTL converter module based on the FTDI chip, the whole thing is solved with one connection fewer. In fact, on the programmer side we will no longer have to short anything, but we will have to exchange the TX and RX contacts with each other as shown in **Figure 8** and in **Table 3**.

The described connections are needed only during the programming phase of the ESP32-CAM. Once the programming is complete, simply power the ESP32-CAM module using only the 5V and GND power wires. It is no longer necessary to connect it directly to the Arduino or make any other wiring.

In the second and final installment of this article, we'll see the procedure for the firmware installation, the correct setup of the camera lens for an optimal focusing, the positioning of the reader on the water meter and, of course, the whole AI-based process for the correct recognition and readout of the counter's elements. Stay tuned! ◀

240213-01

Table 3: FTDI to ESP32-CAM Wiring.

FTDI	ESP32-CAM
RX	U0TXD
TX	U0RXD
5V	5V
GND	GND

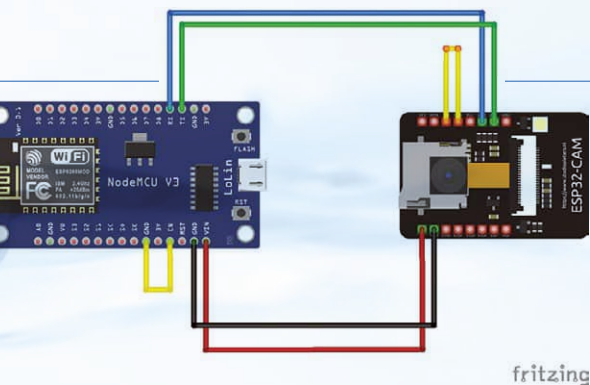


Figure 7: Wiring to program the ESP32-CAM with an ESP8266.

Table 2: ESP8266 to ESP32-CAM Wiring.

ESP8266 ⇒ ESP32-CAM	ESP8266	ESP32-CAM
TX ⇒ U0TXD	EN ⇒ GND	RX ⇒ U0RXD
5V ⇒ 5V		
GND ⇒ GND		

Questions or Comments?

Do you have technical questions or comments about this article? Feel free to write to the editorial team of Elektor at editor@elektor.com.



Related Products

- **ESP32-Cam-CH340 Development Board**
www.elektor.com/19333
- **FTDI Serial TTL RS232 USB Cable**
www.elektor.com/20173

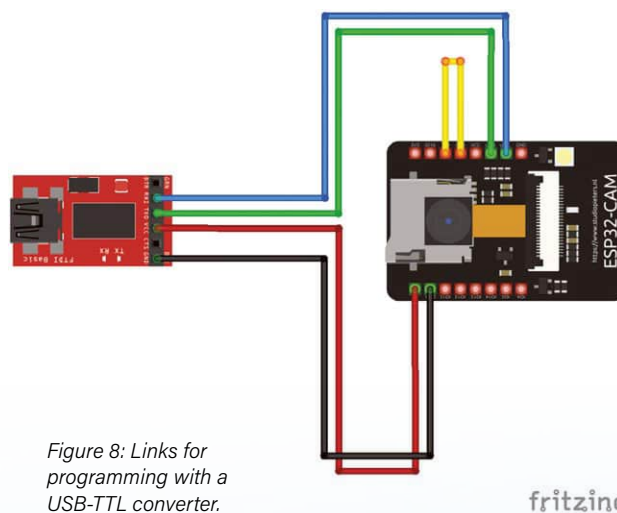


Figure 8: Links for programming with a USB-TTL converter.

A GSM Alarm

Harnessing GSM Technology for Remote Garage Safety

By Pascal Rondane (France)

The article describes a low-power, cost-effective security and fire alarm system for a rented storage space without electricity. It utilizes a GSM transmitter for alerts and achieves around two years of operation without recharging the battery.

I was led to carry out this project following the rental of a box to store my motorbikes and bicycles. The place is located a few kilometers from my home, in the basement of a building without electricity. I wanted to protect my garage against theft and fire, getting an alarm when the door was opened or when my fire detector was triggered. The finished project is shown in **Figure 1**. My system has been running for almost two years without recharging the battery (12 V, 7 Ah). I would like to thank Vincent Ruggieri for his assistance.

Designing the System

The alarms will be sent via a GSM transmitter using a SIM card. Of course, a subscription is needed, but, in France, one of the providers is famous for offering a plan which costs only €2 a month. That is suitable for this project. The system had to have very low current consumption, as the addition of a solar panel was not possible in that basement.

I tried to find a ready-made low-power GSM modem, as I didn't have the time that would be needed for the hardware and software development of a fully custom solution. The goal was also to find a reasonably affordable unit — around €40 in that case.

I went with a GL90 Plus GSM modem. It is easy to source from various sellers on AliExpress and provides the following features:

- Ability to send text messages and phone calls in the event of an alarm
- Ability to send periodic text messages to ensure the proper operation of the GSM transmitter
- Configurable battery status measurement, which enables the sending of a battery-low SMS alarm



Figure 1: The finished project.

- Input voltage from 5 V to 16 V
- Very low standby consumption of less than 4 mA
- Monitoring of 8 configurable NO or NC inputs
- Configuration via text message, so no PC connection is required
- Version with an output allowing you to connect a siren is available
- GSM antenna and coax cable

Building the Alarm

The GL09's inputs are accessible through an HE10-10 connector (10 pins, dual row, 2.54 mm pitch). This connector is on the side of the unit and is made to be used with a flat cable. Also, these inputs are meant to be connected to switches, i.e., they don't tolerate any voltage on them and can be easily destroyed. To address these issues, I designed an adapter board (**Figure 2**) to which the GL09 module can be bolted.

The interface board allows the use of the GL09 with either passive, voltage-free switches (also called dry contacts) or with external sensors



Figure 2: A closer look at the adapter PCB.

that have voltage outputs (for example, 6 to 12 V for a logic High, 0 V for a logic Low). This flexibility is accomplished by using optocouplers that can be switched in and out of the circuit using jumpers.

The switches or sensors that will trigger the alarms are connected to the PCB using pluggable terminal blocks. A jumper is used to switch the module to programming mode (JP14) and a connector is provided for the tamper protection (JP9). Each optocoupler can be configured using two jumpers that are installed on three-terminal male headers, allowing either a dry contact or a voltage signal to be connected. The schematic of each input can be seen in **Figure 3**.

If you want to connect a dry contact, place the jumpers between pins 2 and 3 for both JP2 and JP6. To use a voltage input between 6 and 12 V, place the jumpers between pins 1 and 2. If you want to use passive switches exclusively, you don't need to populate the optocoupler or R2. I still recommend populating the header for JP4, which is useful for testing to simulate alarms during the configuration of the GL09 module. The complete schematic is shown in **Figure 4** (next page), showing seven identical optocoupler blocks as well as two more jumpers, two capacitors, and the connectors.

The PCB (**Figure 5**) is simple, with all SMD components being in the 1206 package and all others being through-hole components. I have routed it with Eagle, and you can find the Gerber at [1]. The entire assembly can be mounted in a watertight electrical junction box.

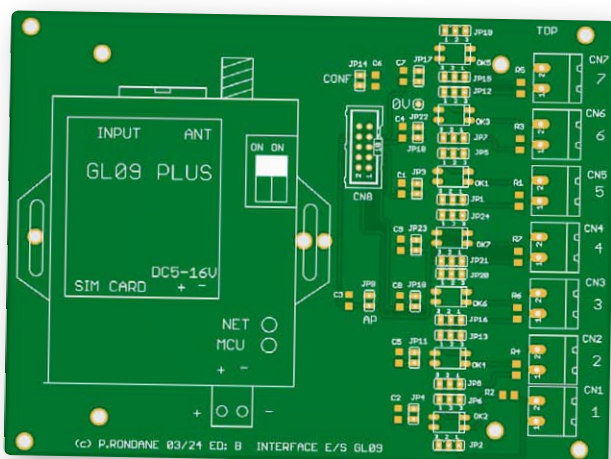


Figure 5: Top view of the adapter PCB.

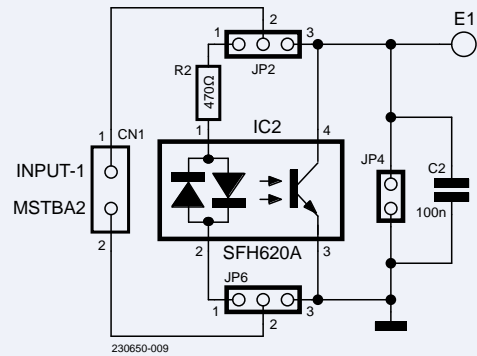


Figure 3: One section of the optocoupler circuits.

Setting Up the Transmitter

The instructions provided with the GL09 module are detailed enough to get you up and running quickly. An alarm text message can be sent to up to six predefined mobile phones when an alarm input is triggered. The alarm text message can be customized with your own wording (e.g. "garage door open") and its return to normal status with another customizable message (e.g. "garage door closed"). Three optional alarm modes can be selected: phone call, text message, or both. Note that during a telephone call, the transmitter does not play any vocal message and disconnects the call as soon as it is answered.

The input power supply voltage can be measured and when it falls below the voltage limit setting (for a 12 V battery, the threshold is 11.6 V) the transmitter sends an alarm text. I've set the transmitter to send an SMS once a day to indicate that it is working properly. One of these daily status reports SMS is shown in **Figure 6**. It includes the date and

time, the current state of the inputs, the bit mask for the "armed" inputs, as well as the current battery voltage and the programmed low-voltage threshold.

To extend battery life, the energy saving operating mode must be selected. In that mode, the power consumption is only approximately 30 μ A at 12 V, according to the manufacturer.

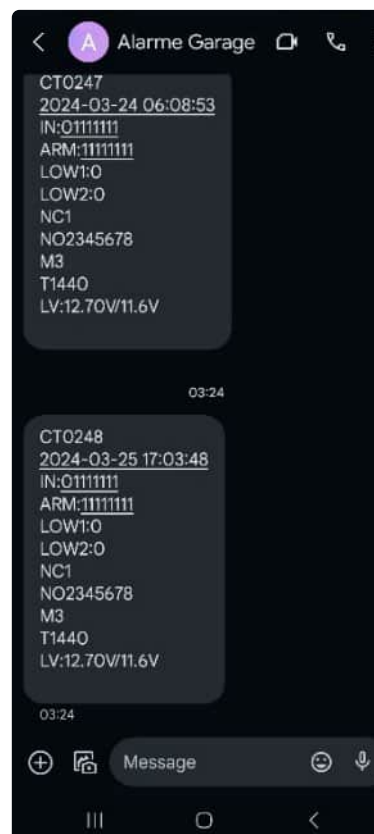


Figure 6: Examples of received SMS messages. *T* is the periodicity of the status reports (here T1440 means 1440 minutes or 24 hours), *M* is the working mode (here M3 means that an alarm triggers both an SMS and a phone call).

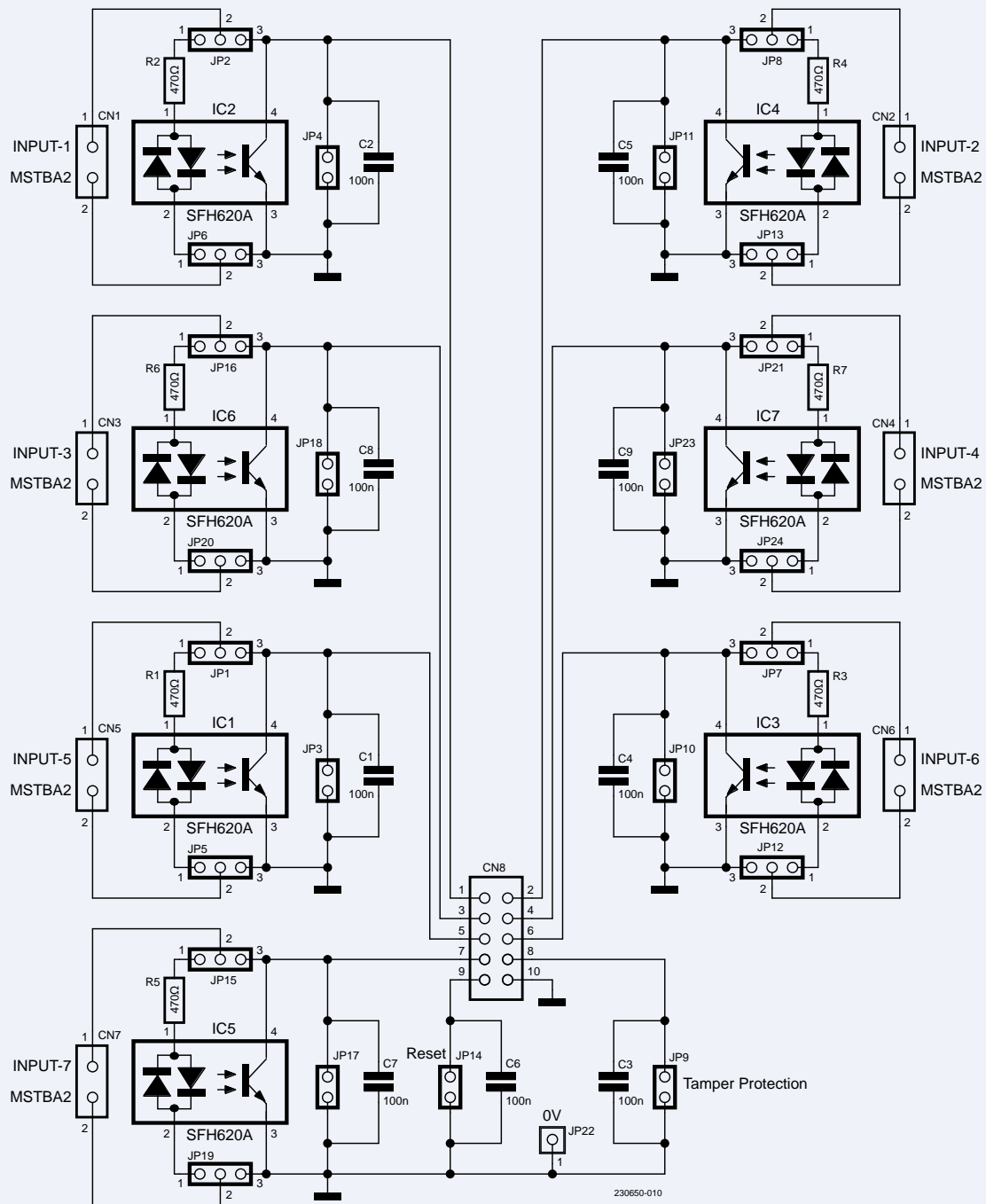


Figure 4: Project schematics.

Fire Detection

I've fitted my garage with one of these standard smoke detectors that use a 9 V battery and are mandatory in houses in many European countries. I've modified the unit to retrieve the signal when a fire is detected. There are two solutions for this. The first one is to detect when the built-in LED of the detector is powered on, by connecting one wire to +9 V from the battery and another wire to one of the LED's series resistor terminals. This signal can be used to drive the LED of one of the optocouplers on the interface board. The second option is available if your fire detector uses the RE46C181 [2] integrated circuit,

which is very common. You can get a voltage of about 8 V between pin 7 (TESTOUT) and ground when smoke is detected, and that voltage can also be used to drive one of the optocouplers described above. [▶](#)

230650-01

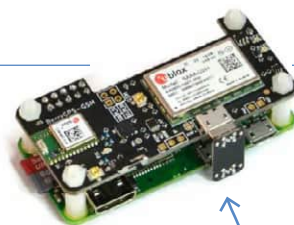
Questions or Comments?

Do you have technical questions or comments about this article? Feel free to contact the author at pascal.tours@gmail.com or Elektor at editor@elektor.com.



About the Author

Pascal Rondane's hobby has been electronics since he was a teenager, and he has owned every issue of Elektor since the beginning of the French edition in 1978. He trained as an electronics technician and worked for 20 years in a company that maintained Motorola radio equipment, and repaired electronic boards for IBM France. He then worked for 22 years in after-sales service and test bench design in a major French group manufacturing road signs. He's been retired for a year, which leaves him more time for his favorite hobby, as well as allowing him to take part in the activities of the *Association du Centre Historique de la Diffusion Radiophonique* (ACHDR), dedicated to the safeguarding of the French audiovisual heritage [3].



Related Products

- > **OzzMaker BerryGPS-GSM for Raspberry Pi**
www.elektor.com/19326
- > **D. Ibrahim, GSM/GPRS Projects Based on PIC Microcontrollers and Arduino, Elektor, 2017 (E-Book)**
www.elektor.com/18203
- > **Crowtail-4G SIM A7670E Module GPS Breakout Board**
www.elektor.com/20542

WEB LINKS

- [1] Related Downloads: <https://elektormagazine.com/230650-01>
- [2] RE46C181 Smoke Detector: <https://microchip.com/en-us/product/RE46C181>
- [3] ACHDR association: <https://achdr.over-blog.com>

JOIN OUR COMMUNITY



Subscribe today at
elektormagazine.com/ezone-24

GET FREE
DOWNLOAD



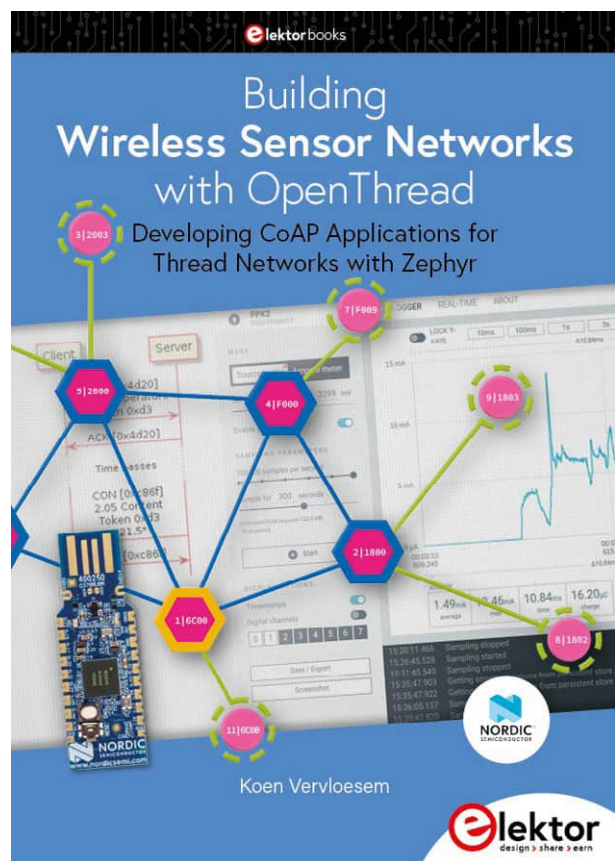
elektor
design > share > earn

Low-Power Thread Devices Optimized and Scrutinized

Low Power ... Low Effort?

By Koen Vervloesem (Belgium)

Elektor's newly released book, *Building Wireless Sensor Networks with OpenThread*, not only presents a massive amount of background information on building and programming your own networks to read sensors of all sorts, it also shows practical applications using a Nordic Semiconductor nRF52840 dongle, which is supplied with the book. Let's examine such a use case to see if it's really a stroll in *Low Power Park* along with Zephyr, CoAP, Wireshark, and Nordic.



Editor's Note. This article is an excerpt from the Elektor book, *Building Wireless Sensor Networks with OpenThread*. This excerpt was formatted and lightly edited to match Elektor Magazine's conventions and page layout. The author and editor are happy to help with queries. Contact details are in the **Questions or Comments?** box.

The fact that Thread is often referred to as a “low-power wireless mesh network” seems to beg for concrete numbers on the power consumption of ... (drumroll) ... Thread applications! This oversight is duly compensated by this article, where you will engage in:

- measuring the power consumption of basic Thread firmware on a Nordic Semiconductor nRF52840 Dongle;
- lowering power consumption by disabling unused hardware;
- lowering power consumption by turning the application into a Sleepy End Device (SED).

To start, flash the basic Thread application (more about it in chapter 5 of the book) to an nRF52840 Dongle (the code is at [4]). After this, confirm that the dongle connects to your Thread network, for instance, by monitoring the transmitted network packets in Wireshark. You're now ready to investigate and optimize the application's power consumption on the dongle.

Measuring Power Consumption with Power Profiler Kit

To optimize your Thread application for reduced power usage, its power consumption must be measurable. One notable hardware kit fitting this purpose is the Power Profiler Kit II (PPK2) from Nordic Semiconductor. This device can measure and optionally supply currents ranging from sub- μA to 1 A, using voltages under 5 V. The device whose current consumption you're measuring (in this case, the nRF52840 Dongle) is called the Device Under Test (DUT).

On the software side, you use the *Power Profiler* application that's part of *nRF Connect for Desktop* [1], Nordic Semiconductor's cross-platform development software. Download the latest package for your platform (Windows/Linux/macOS) and run it. Note: On Linux, first make the downloaded AppImage file executable, for instance, with:

```
chmod +x nrfconnect-4.2.1-x86_64.appimage
```

Then open nRF Connect for Desktop and click on *Install* adjacent to *Power Profiler*.

On the hardware side, prepare the nRF52840 Dongle so it can be powered by an external source through its VDD OUT pin. To accomplish this, you need to cut the PCB track that shorts solder bridge SB2, and you need to solder SB1, as explained in [2]. SB1 and SB2 can be found on the board's underside, as pictured in **Figure 1**. Warning: Once you've modified your nRF52840 Dongle in this manner, it will permanently require external power, even for flashing a new firmware image over USB.

The PPK2 has two modes for measuring power consumption:

- **Ampere Meter mode** — the Device Under Test receives power from an external source (for instance, USB or a battery). You can use this mode to test the dongle's power consumption while running on a 3 V battery.
- **Source Meter mode** — the Device Under Test is powered by the PPK2.

In both modes, you need to know where to find the dongle's VDD OUT and GND pins, see **Figure 2**.

In Ampere Meter mode, connect:

- PPK2 VIN to battery BAT+;
- PPK2 VOUT to dongle VDD OUT;
- PPK2 GND to dongle GND;
- Battery BAT- to dongle GND.

In this setup, the battery powers the nRF52840 Dongle, but with the current flowing through the Power Profiler Kit II, so it can be measured.

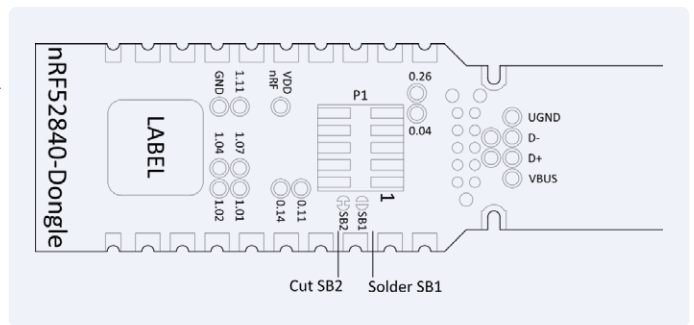


Figure 1: Cut SB2 and solder SB1 on the nRF52840 Dongle to receive power from an external regulated 1.8 to 3.6 V source through VDD OUT. (Image source: Nordic Semiconductor)

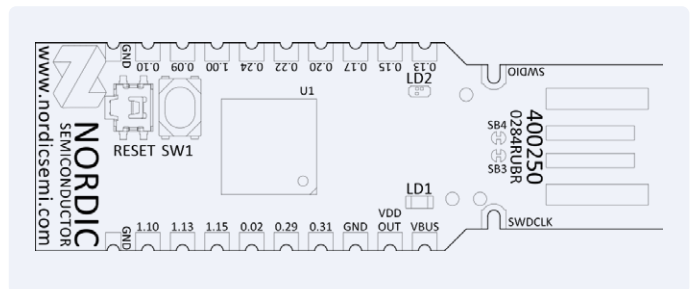


Figure 2: The nRF52840 Dongle's pinout. (Image source: Nordic Semiconductor)

In Source Meter mode, connect:

- PPK2 VOUT to the dongle's VDD OUT;
- PPK2 GND to the dongle's GND.

The connections are shown in **Figure 3**. In this setup, the Power Profiler Kit II provides power to the nRF52840 Dongle and simultaneously measures its current consumption (**Figure 4**).

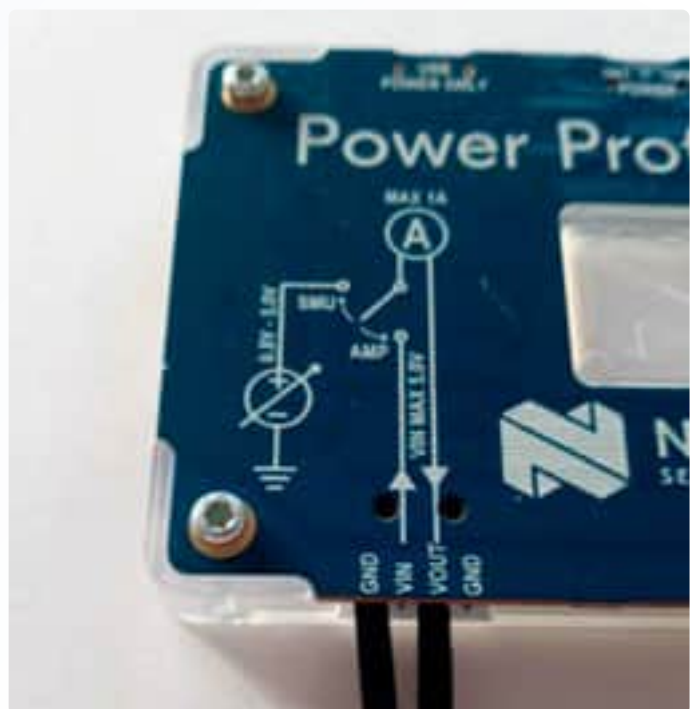


Figure 3: Connect the Power Profiler Kit II to the nRF52840 Dongle in Source Meter mode.



Figure 4: The Power Profiler Kit II ready to measure the current consumption of the nRF52840 Dongle in Source Meter mode.



Figure 5: Connect the Power Profiler Kit II to your PC.

Thread Application's Power Consumption

Next, connect the PPK2 to your computer using a USB cable plugged into the USB DATA/POWER port, and turn the POWER switch beside the USB port to ON as shown in **Figure 5**. Open the Power Profiler app in *nRF Connect for Desktop* and click on *Select Device* at the top left. Choose PPK2. The application may prompt you to program the device. Confirm this by clicking on *Program*.

Choose *Source meter* or *Ampere meter* for the mode, depending on how the PPK2 is connected to your dongle. Set the supply voltage to 3300 mV and turn on *Enable power output* (even in Ampere Meter mode). Immediately after turning on this switch, the dongle will receive the voltage, and you'll see its packets on the Thread network in Wireshark.

Then click on *Start* in the Power Profiler. As illustrated in **Figure 6**, The *Data Logger* tab will now continuously show the device's current consumption.

According to the statistics beneath the graph, the average current consumption is 6.29 mA. This is high, but understandable

considering you haven't applied any power optimizations yet. If you let the Power Profiler run for a while together with Wireshark, you'll observe a higher peak every time the Thread device receives a Mesh Link Establishment advertisement to the `ff02::1` Link-Local All-Nodes address. If you zoom in on this peak, you'll see something not unlike **Figure 7**.

Lowering Power Consumption

A lot of hardware on the dongle that you don't use or don't want to use in production is enabled by default. But every enabled component consumes power! Therefore, let's scrutinize all device tree definitions in:

`~/zephyrproject/zephyr/boards/arm/nrf52840dongle_nrf52840/nrf52840dongle_nrf52840.dts`

and disable the ones you don't need in the project's device tree overlay file:

`boards/nrf52840dongle_nrf52840.overlay:`

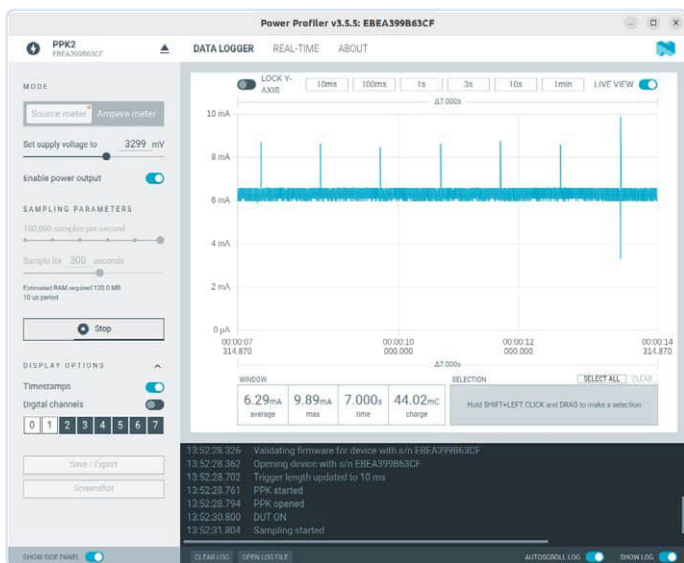


Figure 6: The basic Thread application consumes 6.29 mA on average.

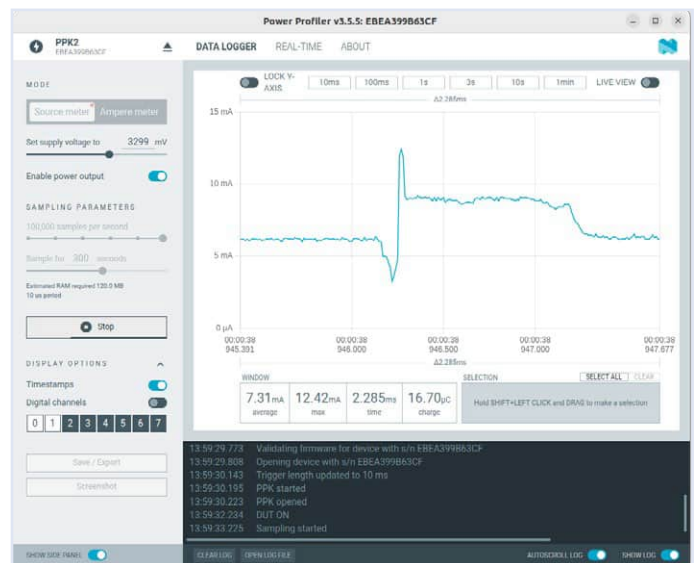


Figure 7: Upon receiving an MLE advertisement, there's a pronounced peak in power consumption.



Listing 1: Device tree definitions.

```
/*
 * Copyright (c) 2024 Koen Vervloesem <koen@
vervloesem.eu>
 *
 * SPDX-License-Identifier: MIT
 */

/*
 * Disabled unused hardware
 */

&adc {
    status = "disabled";
};
&uart0 {
    status = "disabled";
};
&i2c0 {
    status = "disabled";
};
&i2c1 {
    status = "disabled";
};
&pwm0 {
    status = "disabled";
};
&spi0 {
    status = "disabled";
};
&spi1 {
    status = "disabled";
};
&usbd {
    status = "disabled";
};
```



Listing 2: Kconfig configuration file (prj.conf).

```
#
# Copyright (c) 2024 Koen Vervloesem
#
# SPDX-License-Identifier: Apache-2.0
#

# Enable networking and OpenThread
CONFIG_NETWORKING=y
CONFIG_NET_IPV6_NBR_CACHE=n
CONFIG_NET_IPV6_MLD=n
CONFIG_NET_L2_OPENTHREAD=y
CONFIG_OPENTHREAD_THREAD_VERSION_1_3=y
CONFIG_OPENTHREAD_SLAAC=y
CONFIG_OPENTHREAD_PING_SENDER=y
CONFIG_OPENTHREAD_DNS_CLIENT=y
CONFIG_OPENTHREAD_MLR=y

# Kernel options
CONFIG_MAIN_STACK_SIZE=2560

# Enable power management
CONFIG_PM_DEVICE=y

# Create Sleepy End Device
CONFIG_OPENTHREAD_MTD=y
CONFIG_OPENTHREAD_MTD_SED=y
CONFIG_OPENTHREAD_POLL_PERIOD=1000

# Disable USB
CONFIG_USB_DEVICE_STACK=n
CONFIG_BOARD_SERIAL_BACKEND_CDC_ACM=n
```

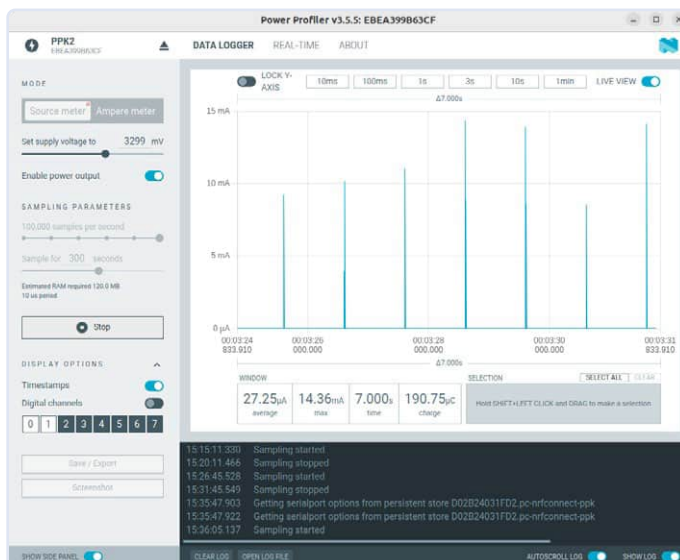


Figure 8: The basic Thread application polling its router as a Sleepy End Device every second is found to consume 27 µA on average.

The program to do so is given in **Listing 1**. **Note:** The Zephyr shell will no longer be accessible over USB when disabling `uart0`.

Additionally, the `Kconfig` configuration file `prj.conf` looks like in **Listing 2**. This essentially enables power management and disables USB. It also lacks the options to enable logging and the Zephyr shell. Additionally, it configures the device as a Sleepy End Device, so the radio can sleep most of the time, and it configures the polling period (the interval between requests from the SED to its router) as 1000 ms.

Now rebuild the firmware:

```
$ west build -p auto -b nrf52840dongle_nrf52840
```

and create the package and flash the firmware to the dongle. If you measure the current consumption now, the average is considerably less at around 27 µA (**Figure 8**). You clearly see a peak of more than 10 mA in current consumption every second, and at


the same time Wireshark shows a data request from the SED to its router. If you send a ping request to the SED, you'll observe that the response can take up to a second, which is the device's polling period (**Figure 9**).

The longer you set the polling period, the lower the current consumption but the less responsive the device becomes. You can test this by adjusting the Kconfig configuration variable `CONFIG_OPENTHREAD_POLL_PERIOD` to 5,000 milliseconds. Rebuild the firmware and flash it to the Thread device. If you then measure its current consumption, it averages about 7.5 μA , with current peaks spaced five seconds apart as illustrated in **Figure 10**. The disadvantage of this extended period is that a ping request to the SED can take up to five seconds, which makes it significantly less responsive.

Summary and Further Exploration

In this article, I showcased how to measure the power consumption of a Thread application running on an nRF52840 Dongle. Thanks to Nordic Semiconductor's *Power Profiler Kit II* and the accompanying *Power Profiler* application in *nRF Connect for Desktop*, you can get substantial insights into your application's power consumption.

As an example, I detailed the power consumption of this book's basic Thread application. You've learned how to lower the power consumption by disabling unneeded hardware and by configuring the device as a Sleepy End Device. You also learned how the polling period impacts the device's power consumption: the longer the polling period, the lower the current consumption, but the less responsive the device becomes.

This article is merely a starting point in your optimization efforts. If you want to get your Thread application's power consumption as low as possible, you'll have to take advantage of Zephyr's power-management API [3]. This allows you to use the power saving features of the SoC and other devices, such as connected sensors. Your code then reads a sensor measurement, sends the data over the Thread network, suspends the sensor, sleeps, wakes up, resumes the sensor, and starts the whole cycle again. 

240225-01

Questions or Comments?

Do you have any questions or comments related to this article? Email the author at koen@vervoesem.eu or Elektor at editor@elektor.com.



About the Author

Koen Vervoesem has been writing for over 20 years on Linux, open-source software, security, home automation, artificial intelligence (AI), programming, and the Internet of Things (IoT). He holds a Master's degree in Computer Science Engineering, a Master's degree in Philosophy, and an LPIC-3 303 Security certificate. He is teaching Linux and Python classes to students aspiring to an Associate degree in Internet of Things.

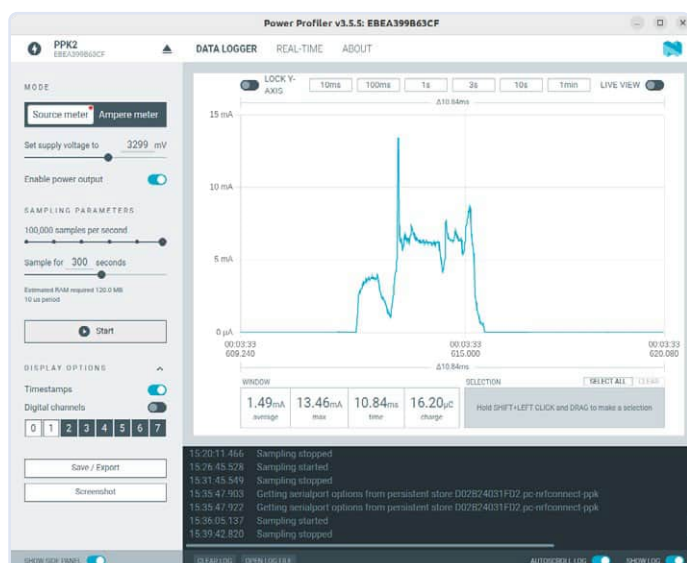


Figure 9: With every data request to its router, the Sleepy End Device experiences a peak in its current consumption.

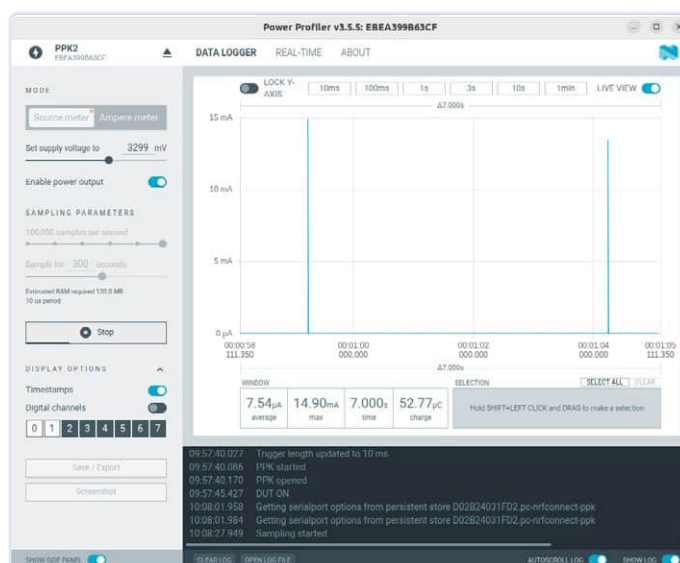


Figure 10: With a polling period of 5 s, the Sleepy End Device is found to use only 7.5 μA on average.

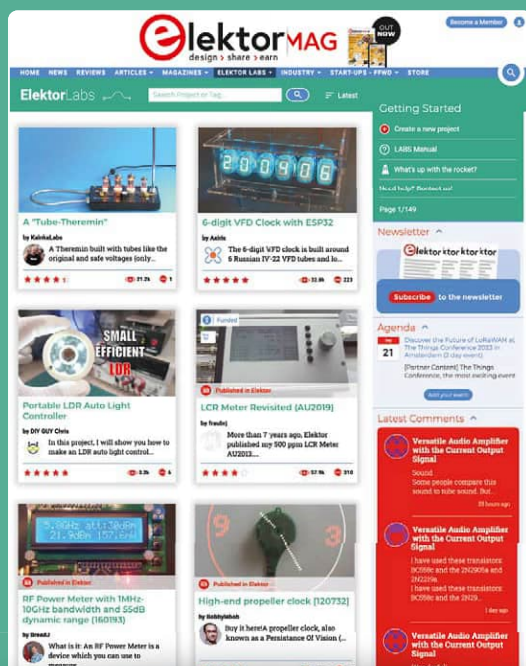


Related Products

- > Koen Vervloesem, *Building Wireless Sensor Networks with OpenThread*, Elektor 2024 (Book and nRF52840 dongle)
www.elektor.com/20860
- > Koen Vervloesem, *Building Wireless Sensor Networks with OpenThread*, Elektor 2024 (E-Book)
www.elektor.com/20861

WEB LINKS

- [1] nRF Connect for Desktop: <https://www.nordicsemi.com/Products/Development-tools/nRF-Connect-for-Desktop>
- [2] nRF52840 Dongle hardware mod: <https://tinyurl.com/rvj5mp47>
- [3] Zephyr's power-management API : <https://docs.zephyrproject.org/latest/services/pm/index.html>
- [4] Main.c, CMakeLists.txt, and configuration overlay for nRF52840 dongle:
<https://github.com/koenvervloesem/openthread-applications>



Share Your Projects Now!
www.elektormagazine.com/e-labs

Ignite Your Electronics Innovations with Elektor Labs

- Free Project Sharing
- Expert Support
- Collaboration Opportunities
- Access to Exclusive Resources
- Get published in Elektor Magazine



elektor
design > share > earn



Marie and Pierre Curie in their laboratory.
(Source: Shutterstock/Morphart Creation)

From Life's Experience

The Gender Gap

By Ilse Joostens (Belgium)

Women and technology still have a somewhat troubled relationship due to a lack of role models, persistent societal prejudices, and stereotypical gender roles. If you carefully observe photos from radio and electronics markets or ham fairs, you will notice that the sedate older man with a receding hairline and a belly is prominently represented.

On the other hand, women are hardly visible. And not only in technology, but also during the annual jamboree for investors and savers of a well-known Flemish business newspaper, it is noticeable that the male part of the population is clearly in the majority.

Fortunately, tech trade shows, hackerspaces, fab labs, and Maker Faires are attracting a more diverse audience these days, but even there, on average, men are still in the majority. However, engineering and financial literacy are not gender-specific at all, and in principle, anyone can delve into them, male or female.

Pimplly Nerds

Throughout history, women have consistently taken on socially important roles, from Viking warriors [1][2] to scientists and engineers. In a world where masculinity is the norm and where men regularly took credit for the work done by women, it seems as if all major

discoveries and technological revolutions were the work of men. Those gentlemen! Had Pierre Curie not stood up for his wife Marie in the nominations for the Nobel Prize in 1903, she would have ended up as a noble unknown, no more than a footnote in history [3].

In astronomy, in the military during World War II and also at NASA, women performed complex calculations manually. Later, when this work was taken over by machines such as the ENIAC, computer programming by nature became a woman's job (**Figure 1**) [4][5]. That changed when, in the late 1960s, employers realized that programming was not just another less-valued administrative job like typing and filing. As the job became more prestigious, more men were trained and were also covertly favored in qualification tests during job applications. As a result, a stereotypical type of man, unkempt, with limited social skills, i.e., the "pimplly nerd," took hold. Even today, IT is still a man's business, with a ratio of about 18% women versus 82% men.



Figure 1: Programming was a woman's job.
(Source: Shutterstock/emkaplin)



Figure 2: Nerd at work. (Source: Shutterstock/
Arsenii Palivoda)



Figure 3: Marthe Douriau (top center).
(Source: forum.retrotechnique.org)

Apparently, the image of the game-addicted obese nerd tinkering with computers in a small attic at night, surrounded by empty pizza cartons, remains stubborn (**Figure 2**).

Within our beloved branch of electronics, women are mostly active in assembly companies where electronic products are built, or PCBs are assembled, but fortunately, there are also more and more female electronics engineers, including a few with their own YouTube channels. Even in the gray past, women were working on electronics at a high level, such as Marthe Douriau (1899–1968) [6], just to name a lesser-known person. This stylish lady with wavy boyish style hair and beaded necklace was not only an engineer at Philips and Ferris but also wrote articles for the magazine *L'Antenne*, where she was a member of the editorial board (**Figure 3**). She also published a series of books on general and automobile electronics, the latter subject admittedly under the pseudonym Marc Dory for fear of not being taken seriously otherwise.

Even now, the technical skills of the average lady are quite questioned when it comes to car technology. A toe-curling offer for a car battery tester on eBay [7] unabashedly states that the device is easy to use and that female drivers can easily use it as well.

Bias the Boss

It's A Man's Man's Man's World, James Brown sang it back in 1966 and unfortunately, it is true. In her book, *Invisible Women*, Caroline Criado Perez shows how the world is largely set up for men and half the world's population is systematically ignored from an (un)conscious human=man-bias. For example, seat belts are not made for women and for years they were only tested with male dummies. With me, the seat belt always cuts into my neck and although a seat belt cover provides some relief, it introduces new annoyances such as the fact that the belt cover tends to shift or, when unbuckled, hinders the belt from being rolled up. Wearing the belt between my breasts is not comfortable either because before you know it, the damn thing is sliding up to my neck again.

Even the tech world, especially in Western Europe, is still a man's bastion, take the Ghent Winter Circus [8] which, under the impulse of a consortium of Ghent entrepreneurs and organizations, is currently being transformed into a technology temple for start-ups and scale-ups. The location looks magnificent and architecturally imposing, but it does not feel safe, secure or warm. That you must first pitch your idea in front of a jury (made up mostly of men) and preferably

do something with AI to get in is the perfect recipe for scaring women away. The same with phrases like "commercial," "stress-resistant," "fanatical" or "result-oriented" in many vacancies for technical positions. When women do choose a technical education or job, many of them eventually drop out. Undervaluation, constantly having to fight against prejudice and an unfriendly or sometimes even unsafe working environment are not foreign to this.

Overcoming existing prejudices and attracting more women to technical jobs such as engineering remains important if we are to move towards a more inclusive world, not primarily designed for men, with innovative ideas for and by women. **◀**

Translated by Hans Adams — 240247-01

WEB LINKS

- [1] Wikipedia: Birka Grave Bj 581: https://en.wikipedia.org/wiki/Birka_grave_Bj_581
- [2] YouTube: Efin Reality — So you want to be a Warrior...: https://youtu.be/_gfo0peYu6o
- [3] Liz Heinecke: When Marie Curie was almost excluded from winning the Nobel Prize: <https://lithub.com/when-marie-curie-was-almost-excluded-from-winning-the-nobel-prize>
- [4] Feminer: Ghislaine Aouragh — Women in tech: are we going back to "the age of the Computer Girls?" [Dutch]: <https://feminer.nl/magazine/rolmodellen/vrouwen-in-tech-interview-chantal-schinkels>
- [5] YouTube: Computer History Archives Project — IBM 701 Rare promo 1953 first of IBM 700 Series Mainframes: <https://youtu.be/fsdLxarwmTk>
- [6] Forum Retrotechnique: Marthe Douriau or Marc Dory?: <https://forum.retrotechnique.org/t/marthe-douriau-ou-marc-dory/79440>
- [7] eBay: Car Battery Tester Analyzer for Automobile: https://ebay.co.uk/itm/294919416437?chn=ps&_ul=GB&mkevt=1&mkcid=28
- [8] Winter circus Ghent: <https://wintercircus.be>

DIY Cloud Chamber

Making Invisible Radiation Visible

By Matthias Rosezky (Austria)

Did you ever want to see radioactivity and ionizing radiation with your own eyes? In this article, I describe what you can build yourself with just a few off-the-shelf core components. With this DIY cloud chamber, you can literally see natural background radiation and even visualize the radioactive decay in some samples.

What's the first thing that comes onto your mind when you hear the terms "radioactivity" or "ionizing radiation?" Probably not many positive things and more like danger, the Chernobyl or Fukushima disasters. However, this kind of radiation has been around for a lot longer than we've been using it and been able to detect it. In fact, natural, so-called background radiation is always all around us and has been for all of humanity's existence. It's nothing out of the ordinary for us, although we may not even be aware of it. In fact, there are two sources for natural radiation: One is terrestrial, which comes from below us in the earth itself, and the other is cosmic, which is all the high-energy radiation coming mostly from outer space. Cosmic radiation is not as important for us as the terrestrial part, though, as this mostly only affects aviation, astronauts, and satellites. The terrestrial part of background radiation is a lot more relevant for us down on Earth. It's being generated by a multitude of naturally occurring radioactive isotopes and they're pretty much everywhere: in rocks, minerals, in the air, in the water, in your food and in your body. Most of these come from

the radioactive decay of uranium and thorium, but there is also a naturally occurring unstable isotope of potassium, for example. In fact, this makes you slightly radioactive as well.

A Window on Radiation

So, you may ask yourself, "If this radiation is all around us, is there a way that we can actually see it with our own eyes?" Sure there is, and it played an important role in particle physics

in the early 20th century: the cloud chamber. With this device, you can not only see ionizing radiation with your own eyes, but you can also differentiate the sort of radiation that's visible. There are two types of cloud chambers; the one I'm focusing on here is called the "diffusion cloud chamber," invented in 1936 by an American physicist. It can be used continuously with virtually no downtime and, with today's technology, can be built pretty easily in your home, which is exactly what I did.

The Cloud Chamber

For a cloud (diffusion) chamber, you essentially need two things: a very cold metal plate at the bottom and some alcohol like isopropyl alcohol above it (**Figure 1**). When the alcohol evaporates, it saturates the air, and

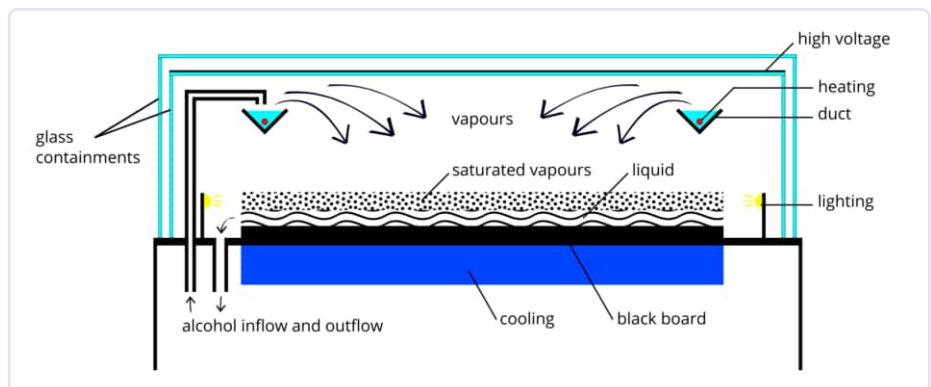
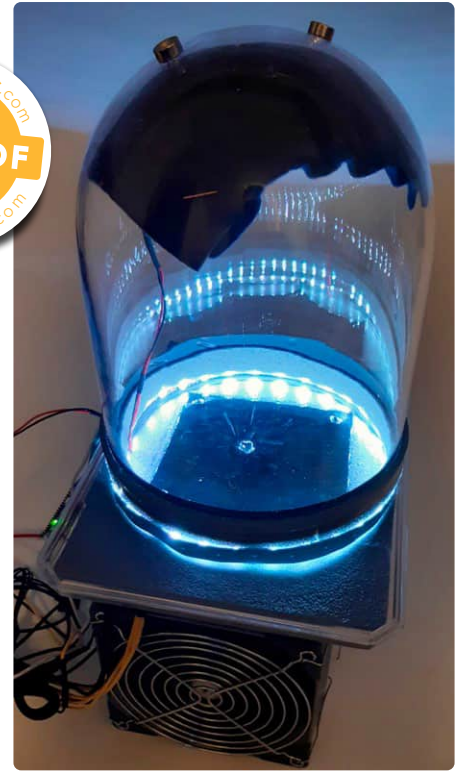


Figure 1: Diagram of a continuous operation cloud chamber. (Source: Nuledo, CC BY-SA 4.0 — <https://creativecommons.org/licenses/by-sa/4.0> — via Wikimedia Commons)

as it cools, it also falls. If this saturated air mixture then comes near the cold plate, the air will condense on the cold plate to form liquid again. However, just above the plate, there's a small section of air where this air-alcohol mixture is supersaturated and on the brink of condensation, just waiting for anything to disturb it. Here is where ionizing radiation comes into play. As the name suggests, it ionizes air molecules when passing through this layer, and these left-behind ions act as the condensation nuclei for the mixture (**Figure 2**). It condenses along the line of ionization and forms a small cloud, therefore revealing where the ionizing particle travelled. Depending on what type of particle went through the cloud chamber, you will see different kinds of trails. Most of what you will be seeing are either alpha or beta particles from their respective decays. The much-heavier alpha particles will leave short, thick trails that appear very straight. In contrast, beta particles leave much longer, thinner, and sometimes quite jagged trails in the cloud chamber (**Figure 3**). You can also insert small radioactive samples into the cloud chamber yourself, and maybe you will see some other decay types too.

Building One Yourself

So, finally, how did I go about building such a chamber from scratch? As I mentioned, the core idea of this cloud chamber is built around a very cold metal sheet and some alcohol vapor. To start with, let's take a look at the metal sheet. To have optimal heat distribution over the entire area, I chose a 4 mm-thick 10×10 cm large copper sheet (**Figure 4**). Copper is an excellent conductor of heat, so the temperature over the whole sheet is very even. This allows me to completely cool down the 10×10 cm area to about -30°C and use it in its entirety to watch for trails, which is helpful when inserting some samples into the cloud chamber. If you'd like, you can also use aluminum, which is a lot cheaper. Keep in mind that it probably won't perform as well, though. To cool it down, I am using a thermoelectric cooler, also known as a Peltier cooler, which is essentially a small heat pump without any moving parts. When applying some voltage to it, one side cools down while the other heats up, maintaining a constant temperature gradient over the two sides.

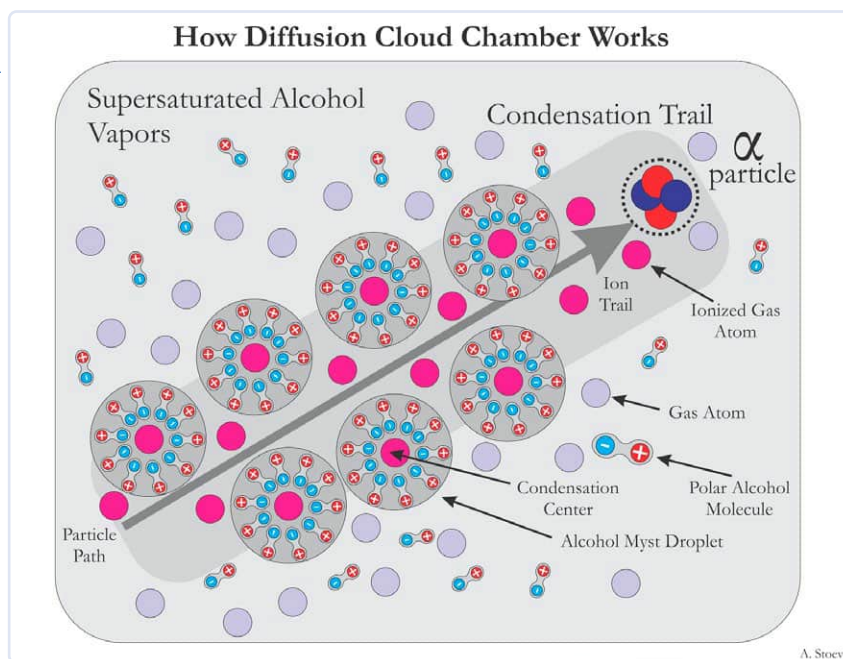


Figure 2: Forming of alcohol condensation trails in Diffusion Cloud Chamber. (Source: Kotarak71, CC BY-SA 4.0 — <https://creativecommons.org/licenses/by-sa/4.0> — via Wikimedia Commons)

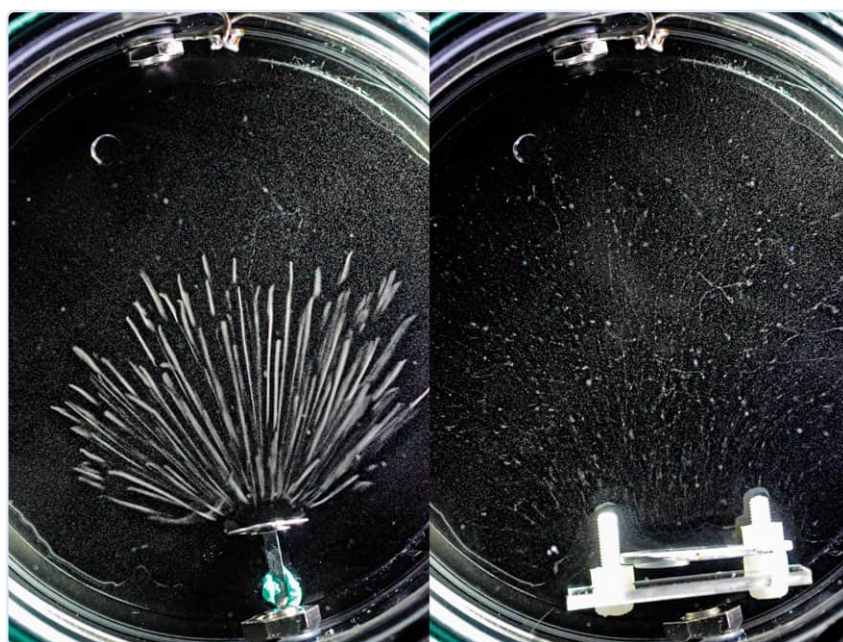


Figure 3: Example image of: (Left) Alpha tracks from Am-241 source. (Right) Beta tracks from Sr-90/Y-90 source. (Source: Kebuk awan, CC BY-SA 4.0 — <https://creativecommons.org/licenses/by-sa/4.0> — via Wikimedia Commons)



Figure 4: Copper sheet on top of the CPU cooler.

The cold side is connected directly to the copper sheet via some thermal paste, and the hot side needs to be cooled as well as possible so that it doesn't heat up much. This is needed because the gradient is constant, so the temperature of the cold side depends on the temperature of the hot side. I'm using a beefy dual tower CPU cooler with two fans normally used in PCs. With conventional air or liquid cooling, you will never be able to go below ambient temperature on the hot side, so the required temperature gradient will be at least around 50°C to reach -30°C if you're using the thermoelectric cooler (and later the cloud chamber) at room temperature. To achieve this, you will need multiple Peltier coolers in series, or you could go straight to using an already cascaded one such as the TEC2-25408. This is a cooler with two single modules stacked internally, so you can use it exactly like a (thicker) single unit, but it is capable of a much higher temperature gradient of up to 80°C (admittedly only in the most ideal of circumstances). 50°C to 60°C with this single module is easily possible if your copper sheet isn't too large.

Overcoming Assembly Challenges

To mount the copper sheet, thermoelectric cooler and air cooler securely together, I wanted to have something that can be screwed and unscrewed in case I needed to change anything. I also wanted to be able to really apply some good mounting pressure to the thermoelectric cooler with all the thermal paste. Because of this, I used the Intel mounting brackets supplied with the CPU cooler, drilled the fitting holes into the copper sheet and used plastic screws and standoffs to connect all the parts together. Now, plastic screws are an obvious compromise, as they aren't nearly as strong and reliable as normal metal screws. However, using metal screws near both the cold copper sheet and the warm CPU cooler is a terrible idea, if only because of the huge heat transfer that would occur between the hot and cold sides. Plastic, on the other hand, has very poor thermal conductivity, which is essential in this case. To further decrease the thermal load on the copper sheet, I also added some thermal insulation to the bottom of it and around the Peltier element, as seen in **Figure 5**.

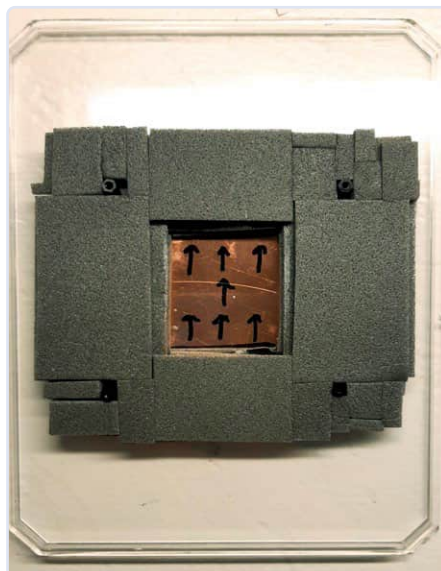


Figure 5: Copper sheet with thermal insulation and plastic standoffs.

To minimize disturbances to the sensitive cloud layer and to make sure that the air mixture saturates with alcohol later on, a bell jar will be placed around the copper sheet, so it needs some space to sit on. For this purpose, I used a simple sheet of acrylic larger than the diameter of the bell jar and mounted it to the copper sheet from below using some plastic standoffs (**Figure 6**). This also provides some space to have some bright LEDs that can shine tangentially onto the copper sheet and illuminate the fine layer of mist. This light helps a lot to make all the trails clearly visible on the copper sheet by increasing the contrast. To further improve visibility, I also covered the exposed top surface of my copper sheet with some black electrical tape.

Power Supply and More

Now, to power all of this, I used a standard ATX power supply that I had lying around from an old PC build. These power supplies can output up to 12 V, which is fed directly into the thermoelectric cooler. Some 5 V lines can be used to power the fans directly without any controller and make them spin silently while providing some airflow to the giant heat sink. In my case, I added a switch between 5 V and 12 V as a crude fan speed toggle. The LEDs that I used are addressable 5 V RGB LEDs, just so that I could experiment with different colors and brightness values. For this reason, I connected them to the power supply too and used a Raspberry Pi Pico to set all the brightness and color values. I ended up just using a slight cold white at maximum



Figure 6: Black copper sheet with the sheet of acrylic around it. I covered the acrylic with some more insulation to provide a soft seal for the bell jar.

brightness, which works the best for me at looking at the intricate trails.

You might have noticed that I haven't talked about the alcohol yet. It needs somewhere to go above the cold plate without dripping. To do that, I took some pieces of felt (**Figure 7**) and mounted them inside on the top of the bell jar with some magnets, as shown in the header picture of this article. These hold it securely enough without requiring any more complicated way of mounting, and with the added benefit of making the pieces easily removable. One final improvement, though, before mounting it all in place, is a small heater for the alcohol. You see, to get an even thicker cloud layer above the cold plate, you could either decrease its temperature (which would require a higher voltage or multiple Peltier coolers)

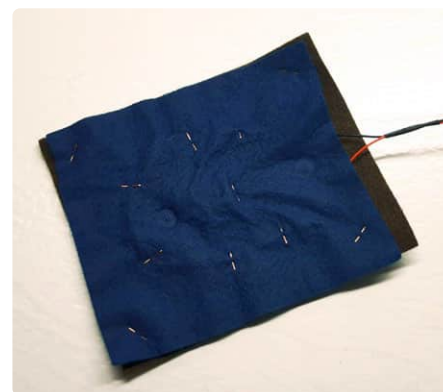


Figure 7: Pieces of felt stapled together with the resistive heater.

or increase the temperature of the alcohol so that more of it evaporates. For this reason, I simply added a couple of resistors to the pieces of felt, stapled them securely in place and connected them to the 12 V rail of my power supply. **For safety reasons, you must be absolutely sure that nothing can short out near the alcohol because it is highly flammable after all!** In my case, the resistors are pretty large in value, only resulting in about 5...10 °C above ambient, but, even with that, you can already see a clear improvement.

Startup and Results

To start the cloud chamber up, I just put some isopropyl alcohol onto the felt so that it is well-saturated, but not dripping. Then I put the bell jar around the top plate and start the cloud chamber up. It will take a couple of minutes to fully cool down to around -30°C. A small puddle of alcohol will form above the plate and prevent ice from forming on the cold plate as well. After that, slowly more and more of a

cloud layer above the plate begins to emerge and soon after this, trails will start developing all over this region.

The results can be seen in this video [1], where I also put a small, slightly radioactive sample into the cloud chamber. Moreover, the project is available on the Elektor Labs project page [2], as well as on Hackaday [3].

That's it! It's not all too complicated and works great for what it's worth. The fact that you can build such a well-working cloud chamber with just a few standard components and some tinkering does not cease to amaze me. Let me know what you think about this project! ◀

230495-01

Questions or Comments?

If you have any technical questions, you can contact the Elektor editorial team by email at editor@elektor.com.

About the Author

Matthias Rosezky is a graduate student in Vienna, Austria. Specializing in radiation physics, he is a tinkerer and maker at heart. He loves programming and working with electronics to create interesting things and improve upon existing designs. In some of his other projects [4], Matthias has taken this philosophy and built affordable open-source radiation-detection hardware.



Related Products

> **MightyOhm Geiger Counter Kit (incl. Case)**

www.elektor.com/18509

> **Electromagnetic Radiation Tester WT3122**

www.elektor.com/20521



WEB LINKS

- [1] Putting Thorianite in my DIY Peltier cloud chamber: <https://youtu.be/BRjhpcfuWA>
- [2] Elektor Labs page for this project: <https://elektormagazine.de/labs/peltier-cloud-chamber>
- [3] The project on Hackaday: <https://hackaday.io/project/192146-peltier-cloud-chamber>
- [4] More projects by this author: <https://nuclearphoenix.xyz>

YOUR KEY TO CELLULAR TECHNOLOGY



**WURTH
ELEKTRONIK**
MORE THAN
YOU EXPECT

WE are here for you!

Join our free webinars on:
www.we-online.com/webinars

Adrastea-I is a Cellular Module with High Performance, Ultra-Low Power Consumption, Multi-Band LTE-M and NB-IoT Module.

Despite its compact size, the module has integrated GNSS, integrated ARM Cortex M4 and 1MB Flash reserved for user application development. The module is based on the high-performance Sony Altair ALT1250 chipset. The Adrastea-I module, certified by Deutsche Telekom, enables rapid integration into end products without additional industry-specific certification (GCF) or operator approval. Provided that a Deutsche Telekom IoT connectivity (SIM card) is used. For all other operators the module offers the industry-specific certification (GCF) already.

www.we-online.com/gocellular

- Small form factor
- Long range / worldwide coverage
- Security and encryption
- Multi-band support

#GOCELLULAR

SparkFun Thing Plus Matter

A Versatile Matter-Based IoT Development Board

By Saad Imtiaz (Elektor)

The SparkFun Thing Plus Matter MGM240P makes a striking entrance in the IoT and smart home arena, packed with robust features for developers and hobbyists alike.

Formerly known as Project CHIP (Connected Home over IP), Matter [1] is a protocol developed to enable interoperability among smart home and IoT devices, making the SparkFun Thing Plus Matter board [2] an exciting addition to the IoT development landscape.

The board (**Figure 1**) features a compact 5.84×22.9 cm design, compatible with the Thing Plus (Feather-compatible) form factor. You can develop software for the MGM240P using the Simplicity Studio debugging tool, which is compatible with Windows, Mac OSX, and Ubuntu. The hookup guide [3] provides detailed hardware information and a step-by-step guide to help you get started with the Simplicity Studio IDE.



Figure 1: SparkFun Thing Plus Matter MGM240P.

These are the highlights of SparkFun Thing Plus Matter MGM240P:

- MGM240P wireless module [4]
- Wireless — 802.15.4 wireless protocols (Zigbee and Open Thread) and Bluetooth Low Energy 5.3; Matter-ready
- Silicon Labs EFR32MG24 SoC [5]
- Memory & Storage — 1,536 KB flash memory, 256 KB RAM
- Storage — microSD card socket
- Two rows of headers with 21×GPIO pins
- 4-pin JST Qwiic connector
- EFM32GG12B410F1024GL120-A microcontroller used as a J-Link programmer and debugging IC
- Unpopulated Mini Simplicity connector to connect an external debugger
- 2-Pin JST connector for a single LiPo battery (not included) with and MC73831 LiPo charger and a MAX17048 LiPo cell fuel gauge
- Power consumption — 15 µA when the MGM240P is in Low Power mode

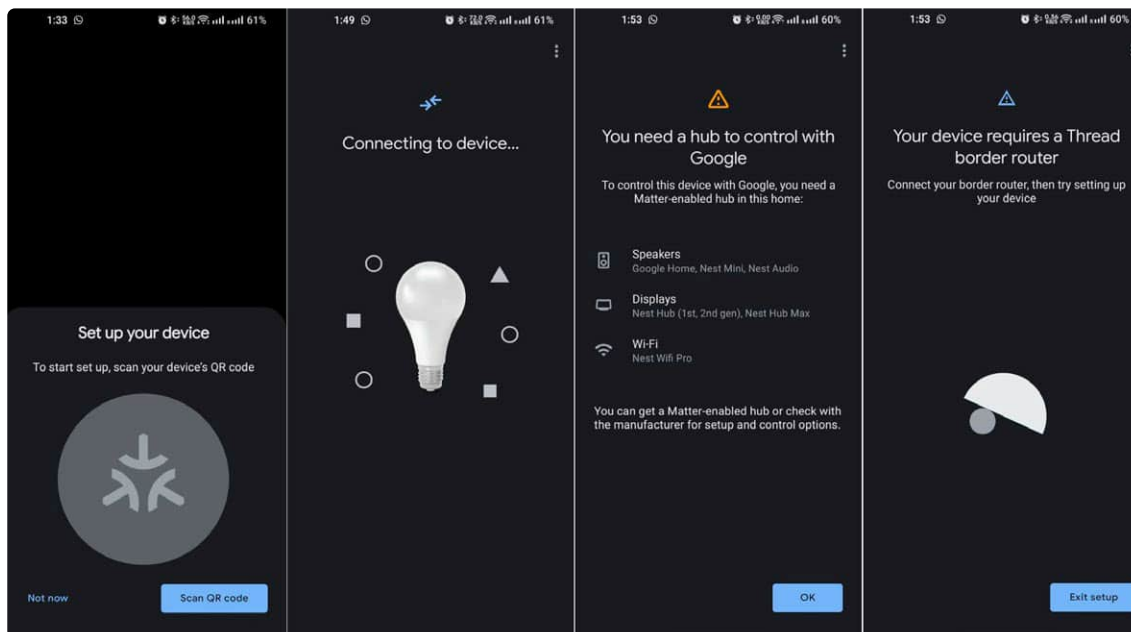
Matter Protocol Support

The board is designed with Matter protocol support in mind, which is a significant advantage for those looking to create smart home or IoT devices that seamlessly communicate with other Matter-compatible devices. The protocol's ability to unify different IoT ecosystems is a game-changer for IoT development.

Wireless Connectivity and Testing

This board is equipped with an MGM12P module, which is based on the EFR32MG12 family from Silicon Labs. This module provides support for multiple wireless protocols, including Bluetooth Low Energy (BLE) and IEEE 802.15.4 (the foundation of Thread and Matter) (**Figure 2**). There are numerous code examples for this

Figure 5: To set up the board with your Google Home app, your phone detects the device shortly after the code is flashed. However, controlling Matter devices with Google requires a Google Hub.



board available on the SparkFun's Simplicity Studio. In my testing, I checked some of these examples, which also include the *Matter Light Over Thread* example, which basically is a Matter lighting app.

Flashing the code onto this board is simple: You first install Simplicity Studio, plug your board in, and the board gets detected automatically. Make sure you upgrade the board's firmware before using it, as it is always good to keep the board up to date to avoid any bugs and problems which were unfixed in previous releases (Figure 3).

After updating the firmware, you can simply click *Create New Project*, and then you have a ton of examples suitable for most of the applications for this board. I selected the example *Matter Light Over Thread*, built the firmware, and uploaded it to the board (Figure 4).

Setting up a Google Home Developer [6] account is required. When you upload the code to your board, your phone will automatically recognize it as a Matter device and ask you to pair. I was able to follow the instructions available on the SparkFun page [7], but unfortunately you need to have a Google Hub to control Matter devices with Google. As shown in the

Figure 5, it says, "Your device requires a Thread Border Router." Regardless, I didn't worry about it, and if I were to have a hub, I'm very confident in saying that it's going to work.

Flexible Power Options

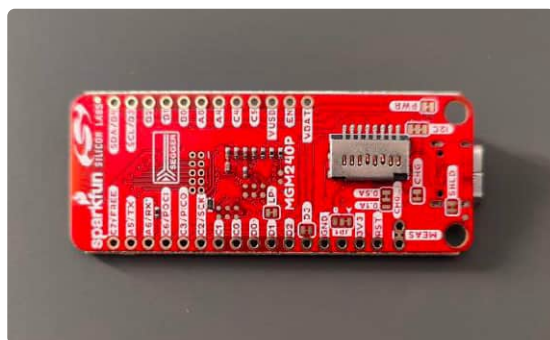
The board (Figure 6) offers flexible power options, allowing you to power it via USB, a Li-Po battery, or an external power supply. It has PTH pins connected to the 3.3 V, V_{USB}, and V_{BATT} nets. The board is designed with a 2-pin JST connector, providing a connection point for a single-cell LiPo battery, making it suitable for battery-powered applications. To ensure stable operation, the input voltage is regulated by a 3.3 V voltage regulator. Additionally, the board incorporates two key components: the MCP73831 Single-Cell LiPo Charge IC, which handles recharging the connected battery when plugged in via USB-C, and the MAX17048 single-cell fuel gauge, which continuously monitors the battery's charge level. In my testing, this feature works as intended. It's a good feature to have on this board as most of the IoT devices should be wireless to be more mobile.

By default, the charge current is configured at 500 mA. However, there's a three-way jumper labelled CHG, allowing users to switch between 500 mA charge current, 100 mA charge current, or disabling the charge IC entirely when it's not needed. In my testing, the default charging circuit was working as intended without any issues. This feature also allows you to slim your project down, as you don't need to include a charging module.

Expansion Possibilities With Qwiic Connectors

One of the standout features of this board is its compatibility with Qwiic connectors. Qwiic is SparkFun's plug-and-play system for connecting various

Figure 6: The board supports multiple power options. It includes features that enhance mobility for IoT devices.



sensors and peripherals (as seen in **Figure 7**). It simplifies the hardware setup process, making it easy to add additional components to your IoT projects.

Debugger

This board is equipped with the EFM32GG12B-410F1024GL120-A microcontroller, serving as a J-Link programmer and debugger. It comes with a Mini Simplicity Connector for those who want to use an external debugger. By default, the board has the debugger set in standard mode, with the debugger WAKE pin connected to V_USB via the LP jumper. However, users have the option to disconnect this jumper to switch the debugger into Low Power mode.

This debugger offers powerful low-level debugging capabilities when paired with Simplicity Studio's debugging tool. You can perform a wide range of standard debugging tasks, including debugger output, setting code breakpoints, and even delving into assembly code for more detailed analysis.

Interoperability

One of the primary goals of the Matter protocol is to provide interoperability between various IoT devices, regardless of their manufacturers. With the SparkFun Thing Plus Matter board, you're in an excellent position to create devices that seamlessly integrate with existing and future Matter-based IoT ecosystems.

Customization and Prototyping

For those who want to customize their IoT solutions, the board offers opportunities for soldering headers and configuring various settings. Additionally, SparkFun's Qwiic ecosystem provides an array of sensors and peripherals to enhance your projects.

Future Project Ideas

With its extensive features and support for the Matter protocol, the SparkFun Thing Plus Matter (MGM240P) is ideal for creating various IoT devices. Explore project ideas, from smart home devices to environmental sensors, and leverage the board's capabilities to bring your innovative ideas to life.

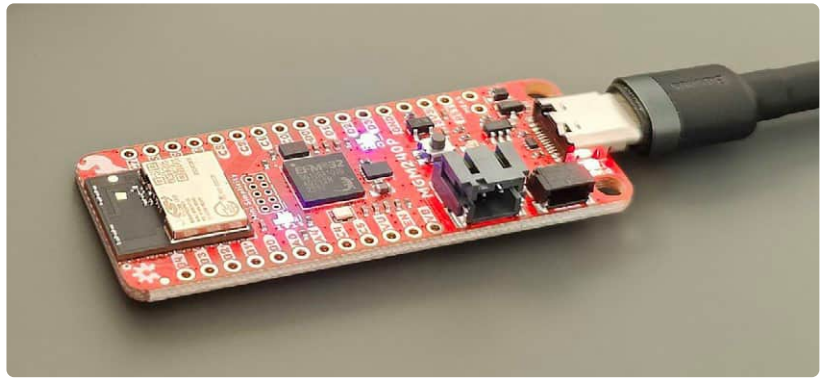


Figure 7: This board stands out for its Qwiic connector compatibility, simplifying the process of adding components to your IoT projects.

The SparkFun Thing Plus Matter (MGM240P) is a good IoT development board tailored for Matter-based IoT solutions. Its compatibility with multiple wireless protocols, Qwiic connectors, its battery-charging feature and extensive resources from SparkFun and Silicon Labs all empower developers to tackle diverse IoT challenges. Whether you're initiating a new IoT project or elevating an existing one, this board serves as an excellent foundation for your IoT development journey. ◀

240251-01

Questions or Comments?

If you have questions about this article, feel free to email the author at saad.imtiaz@elektor.com or the Elektor editorial team at editor@elektor.com.



Related Products

▶ **SparkFun Thing Plus Matter (MGM240P)**
www.elektor.com/20442



Watch the webinar,
"A Matter of Collaboration," to learn how to
develop with the Thing Plus Matter Board
and Simplicity Studio.
<https://youtu.be/vyL0OyRNj1Q>



WEB LINKS

- [1] What Is Matter?: <https://developers.home.google.com/matter/overview>
- [2] SparkFun Thing Plus Matter (MGM240P): <https://elektor.com/20442>
- [3] MGM240P hookup guide: <https://tinyurl.com/mgm240phookup>
- [4] MGM240P wireless module: <https://tinyurl.com/mgm240modules>
- [5] Silicon Labs BG24 and MG24 announcement: <https://tinyurl.com/bg24mg24news>
- [6] Google Home Developer Console: <https://console.home.google.com/projects>
- [7] Tutorial on connecting Thing Plus Matter to Google Nest Hub: <https://tinyurl.com/thing-plus-matter-to-google>

IoT Retrofitting

Making RS-232 Devices Fit for Industry 4.0

By Matthias Lay (Würth Elektronik eiSos)

In industrial environments, you often find older machines that still work reliably but unfortunately do not have an interface to an industrial bus system. IoT retrofitting can be an elegant solution here, as the example of an automatic welding machine with an RS-232 interface shows.

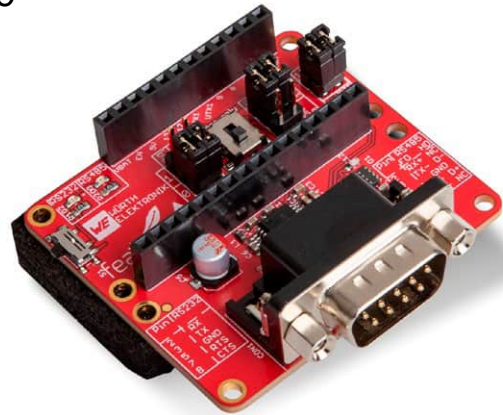


Figure 1: Serial Bridge FeatherWing from Würth Elektronik for the RS-232, RS-422 and RS-485.

Increasing digitalization in industry, often referred to synonymously as Industry 4.0, and the spread of the Industrial Internet of Things (IIoT) require ever more advanced networking of machines in production facilities. The advantages of networking in the industrial environment are already well-known [1] and the importance of the topic for the future success of companies is undisputed. Nevertheless, there are always difficulties with the actual implementation. According to a survey by the Vogel Communications Group, the interface problem is one of the challenges faced when implementing IIoT projects. It describes the problem that production machines are unable to communicate with other systems due to a lack of standardization of protocols and interfaces, or the complete absence of these. One approach to solving this problem is to follow the Greenfield IIoT strategy and replace all machines with new IIoT-capable machines with standardized interfaces. However, this is obviously neither ecologically nor economically reasonable [2].

Rapid Prototyping With Feather Modules

Feather and FeatherWing modules are ideal for rapid prototyping in retrofit-

ting projects. The evaluation boards have a modular design and can be plugged together. Due to the fixed pinning of the Adafruit Feather platform and the possibility of stacking several boards on top of each other, they offer the possibility of adding further functions and interfaces with any microcontroller. This means that functional hardware prototypes can be built, and various configurations evaluated within a very short time.

Würth Elektronik offers a range of such evaluation boards — open-source and fully compatible with the Feather form factor. In addition to the use of sensors and various radio protocols (Wi-Fi/mobile radio) and the operation of the prototype with different supply voltages, these also enable the addition of industrial interfaces. There is a GitHub repository [3] for all open-source boards, including their schematics, BoMs, software, and cloud connectivity descriptions for Azure and AWS.

Reading Out the RS-232 Interface

In retrofitting projects, the use of existing, non-IIoT-capable machine interfaces is particularly advantageous. This allows

machine parameters to be recorded and analyzed without the need to retrofit external sensors. One of the most frequently used interfaces on existing machines in an industrial environment is the RS-232 interface. This is a very robust and yet, one of the most uncomplicated communication interfaces. A basic understanding of baud rate as well as data, parity and stop bits usually enables successful data transmission.

To connect an RS-232 interface to the retrofit prototype's microcontroller, a Serial Bridge FeatherWing from Würth Elektronik (**Figure 1**) is used. This converts a serial interface into a UART interface, which can be used for microcontrollers. By configuring the microcontroller's UART interface in software, one can map common communication protocols, such as Modbus or ASCII, and, by configuring the serial bridge, the physical communication standards RS-232, RS-422 and RS-485 can be mapped in half- and full-duplex operating modes.

IoT Connection via Wi-Fi/MQTT

A suitable IoT interface is required to integrate retrofit prototypes into an existing IoT system. The MQTT protocol can be used as the protocol for the interface. This is

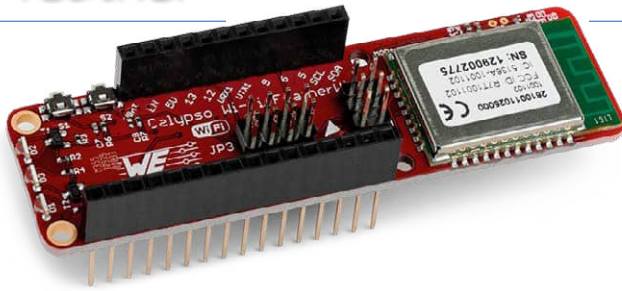


Figure 2: Wi-Fi wireless communication is easy to implement with the Calypso Wi-Fi FeatherWing from Würth Elektronik.

one of the most widely used protocols in the IoT context. Due to its simplicity and low bandwidth requirements, it is particularly suitable for receiving data from many different machines [4]. The advantage is that it can be based on any TCP communication.

The Feather ecosystem allows different transmission methods to be used, depending on the retrofit requirements: For example, a wired transmission, by using an Ethernet FeatherWing or a wireless transmission using a radio module from the FeatherWing family.

As the prototype needed to be implemented and tested as quickly as possible, the Calypso Wi-Fi FeatherWing from Würth Elektronik was chosen (Figure 2). This can easily be integrated into a production facility's existing local network via existing access points (Figure 3) and this eliminates the need to install additional Ethernet cables.

Implementation

The digitization of the production line's welding unit serves as a practical example of the implementation of a prototype (Figure 4). It was decided in advance that this part of the production line was the most suitable for retrofitting. The basic idea is that, by evaluating the welding parameters, conclusions can be drawn about the thermode's current condition. This information can provide maintenance

personnel with indications as to whether a thermode is at risk of reaching its operating limit during the next production shift. In this way, the thermode can be replaced before the actual failure occurs, to prevent unplanned production line downtime.

The Serial Bridge FeatherWing is configured in RS-232 mode according to the machine to be retrofitted and adapted to the welding inverter's baud rate and configuration on the software side. After each weld, the inverter transmits all the parameters recorded for the spot weld using the ASCII protocol.

The welding data is already transferred within the machine via RS-232 from the welding inverter to the PLC in order to create a local quality assurance report. In order to read this data transmission, the RS-232 receiver signal on the PLC side and the RS-232 reference potential must be fed out on the hardware side. These extracted signals and the reference potential are now connected to the Serial Bridge FeatherWing. The prototype now continuously reads the RS-232 transmissions and converts the transmitted data into the data format required by the IoT system.

The microcontroller then forwards the data to the Calypso Wi-Fi FeatherWing. This is connected to the local IoT network and can therefore feed the data directly into the IoT system.



Figure 4: Machine with welding module for the retrofit.

The data now provided in the IoT system (Figure 5) can be used to gain real-time insight into the welding parameters and utilization of the production line and, in the next step, to perform data analysis based on the welding parameters.

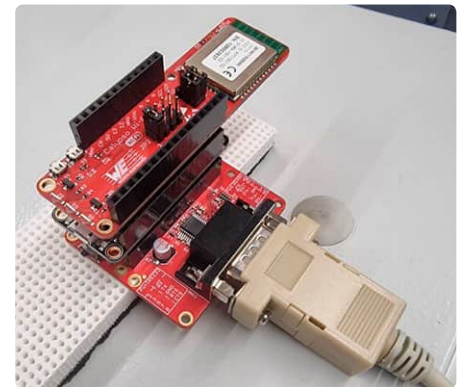


Figure 5: Stacked Feather modules. Top: Calypso Wi-Fi FeatherWing. Middle: M0 microcontroller. Bottom: Serial Bridge FeatherWing.

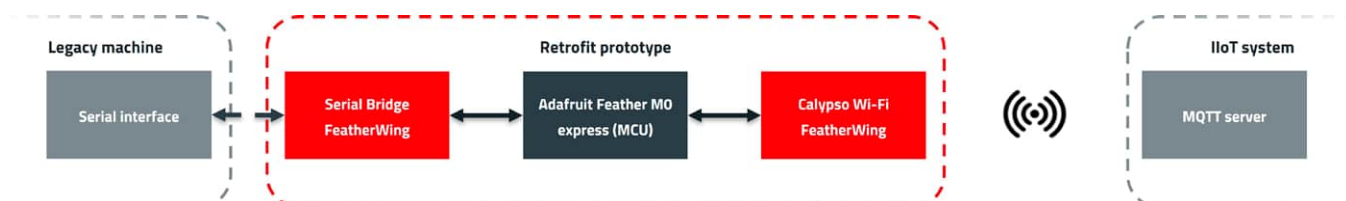


Figure 3: Data flow diagram of the legacy machine's connection to the existing IIoT system.

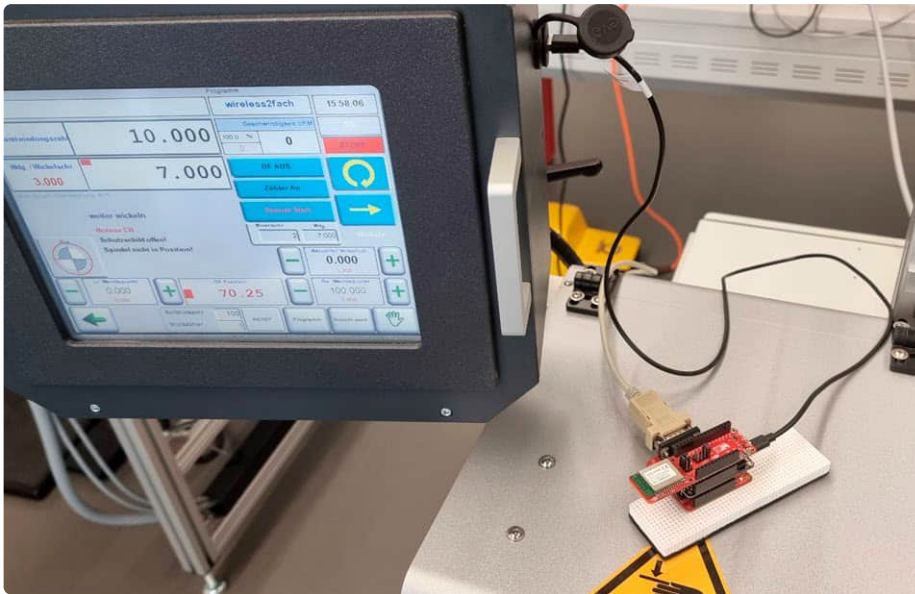


Figure 6: Identical prototype as a retrofit on a winding machine.

Process Improvement Through Analysis

The prototype works and enables continuous monitoring of machine utilization, cycle time, and welding parameters. This also makes it possible to estimate the condition of the thermode.

Analyzing the recorded data has revealed that the adjustment of some welding parameters had beneficial effects on the thermode's service life. For example, it became clear that by reducing the welding power and increasing the welding duration, a significantly longer thermode service life can be achieved with only a minimal increase in cycle time.

However, the retrofit prototype's implementation was not without its problems. As the welding inverter is housed in the base of the

machine and this base is completely covered with metal panels, it acts as a Faraday cage. The prototype therefore had to be mounted outside the base to ensure a stable wireless connection to the access point.

In addition, the current setup only allows data to flow in one direction, namely from the welding inverter to the IoT system. This is because RS-232 communication only uses two data lines, and there can only be one active transmitter per line. To be able to make changes to the welding parameters using the IoT system in the future, communication must be bidirectional. For this purpose, an extension of the prototype with an additional Serial Bridge FeatherWing is being considered. This would make it possible not only to read the communication, but also to send parameter adjustments to the welding inverter.

The prototype presented using a welding inverter can be used in various ways: One example of this is the identical prototype as a retrofit on a winding machine (**Figure 6**).

FeatherWing as a Construction Kit for Prototyping

In conclusion, it can be said that the use of the Feather form factor saves a massive amount of development time when creating the hardware prototype. Instead of designing a board with all the required components, assembling it and putting it into operation, the evaluation boards in Feather format are simply ordered according to the required function and plugged together. This means that software development can begin very quickly and the hardware prototype can be adapted much more easily in the event of unforeseen problems or changes in requirements. In short, the use of the Feather form factor makes hardware development more agile and significantly speeds up the entire prototyping process. ◀

240280-01



About the Author

Matthias Lay studied electrical engineering at Heilbronn University of Applied Sciences, majoring in automation technology and graduating with an MSc. He has been employed at Würth Elektronik eiSos since 2019, initially in the areas of hardware and software development. Since 2023, he has been working there as an IoT system engineer with a focus on IIoT and retrofitting.

WEB LINKS

- [1] Sniderman, B. et. al., "Industry 4.0 and manufacturing ecosystems — Exploring the world of connected enterprises," Deloitte: <https://tinyurl.com/5745usv8>
- [2] Pietrangeli, I. et al., "Smart Retrofit: An Innovative and Sustainable Solution," MDPI: <https://mdpi.com/2075-1702/11/5/523>
- [3] GitHub Repository: <https://github.com/WurthElektronik/FeatherWings>
- [4] Ries, U., "MQTT-Protokoll: IoT-Kommunikation von Reaktoren und Gefängnissen öffentlich einsehbar," Heise Security [German]: <https://tinyurl.com/388yrykb>

Enabling IoT with 8-Bit MCUs

By Joshua Bowen (Microchip Technology)

Since their inception in the 1970s, microcontrollers (MCUs) have been integral in managing various automotive, consumer, and industrial products. Their scope has now expanded to include portable wireless devices and wearable Internet of Things (IoT) applications. Additionally, the healthcare sector has experienced significant growth, integrating 8-bit MCUs into numerous embedded designs.

Embedded electronics utilizing 8-bit MCUs need to be both capable and economically viable, with production scales often reaching hundreds of thousands or even millions of units per application. In automotive contexts, 8-bit MCUs manage numerous subsystems, from motorized seats and windows to smart door handles and tire pressure sensors. This large-scale usage makes even minor cost differences crucial. The maintenance costs of millions of devices, often overlooked during design, can be mitigated by enhancing reliability and durability through code simplification and hardware improvements, reducing the need for software redundancies.

The enduring popularity of 8-bit MCUs is largely due to their continuous innovation in memory, power consumption, packaging, and core-independent peripherals (CIPs).

Significant Enhancements in 8-Bit

With the rise of IoT and the transformation of cities with smart devices, scalable intelligence has become vital for many industries. Upgrades such as smart streetlights and individual parking space detectors require MCUs capable of data collection, processing, and communication. Often, these tasks can be managed by an 8-bit MCU with an on-chip analog-to-digital converter (ADC), while the core remains in a low-power state. This approach is suitable for smart parking garages, connected streetlights and automated urban gardening, where power efficiency is critical. The advantages of smaller devices are evident in their reduced power consumption and compact size, fitting into limited spaces in portable IoT products.

Newer microcontrollers are designed with cost-effectiveness in mind, providing the necessary functionality while being mindful of the user's budget. Additionally, advances in memory technology have significantly increased the capabilities of modern MCUs.

Memory

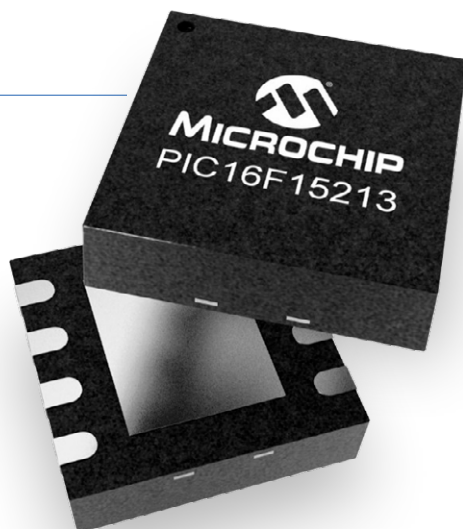
Modern microcontrollers have evolved significantly, driven by advancements in Flash memory. Today's applications demand more complex programs, necessitating increased memory capacity in MCUs. Embedded flash memory in these devices is durable, capable of withstanding rigorous automotive testing and numerous write-and-erase cycles. Current 8-bit microcontrollers offer memory ranging from 384 bits to over 128 KB, catering to a wide range of applications.

Power Consumption

Power efficiency has become a primary focus for 8-bit MCUs, especially in battery-powered applications. For instance, the nanoWatt XLP eXtreme Low Power [1] PIC MCUs feature system supervisory circuits tailored for minimal power usage. These innovations result in the industry's lowest current draws for both *Run* and *Sleep* modes, where devices spend 90 to 99% of their time. Technologies such as *Peripheral Module Disable* further enhance power savings by completely removing peripherals from the power rail and clock tree. Benefits of nanoWatt XLP technology include:

- Sleep currents below 20 nA
- Brownout reset down to 45 nA
- Watchdog timer down to 220 nA
- Real-time clock/calendar down to 470 nA

Figure 1: Many new PIC and AVR product families offer a variety of package offerings as small as 3x3 mm VQFN devices for space constrained applications.



- Run currents down to 50 μ A/MHz
- Full analog and self-write capability down to 1.8 V

Since many 8-bit MCUs are used in battery-powered applications, additional power savings are possible through optimized peripherals, which will be discussed later in this article.

Packaging

A key distinguishing factor of 8-bit MCUs is their ability to fit into small packages, ideal for space-constrained applications in wireless, portable, and wearable products. Packages such as the 8-pin SOIC or 8-pin DFN and the popular 20-pin very thin quad

flat-pack no-leads (VQFN) offer compact solutions (**Figure 1**). For more complex applications, larger packages such as 40-pin PDIP and 44-pin TQFP are available.

Core-Independent Peripherals

Core-independent peripherals (CIPs) enhance MCU functionality by operating autonomously from the core, which is beneficial for low-power and low-cost designs. CIPs can handle various tasks independently, reducing the need for central processing unit (CPU) intervention and thereby increasing system efficiency and reliability. This modular approach simplifies the implementation of touch interfaces, sensor data processing, and more.

This design approach provides a prepackaged means of programming events based on peripherals. For example, the *Event System* can trigger events based on general-purpose input/outputs (GPIO) or program interrupts on multiple channels.

Currently available CIPs for 8-bit PIC and AVR microcontrollers in **Figure 2** are shown color-coded by

8-bit PIC® and AVR® Microcontrollers				
CPU		Memory		
8-/10-/12-bit ADC	(Enhanced) Capture/Compare/ PWM	Input Capture	Direct Memory Access Controller	Configurable Custom Logic
ADC with Gain Stage	Complementary Output Generator	Angular Timer	High Endurance Flash (Data)	Configurable Logic Cell
ADC with Computation*	Complementary Waveform Generator	Charge Time Measurement	Event System	Crypto Engine AES/DES
Comparators	Data Signal Modulator	RTC/IC	IDLE & DOZE	CAN
DAC	Numerically Ctrl'd Oscillator	Signal Measurement Timer	Peripheral Module Disable	(E)USART
High Speed Comparators*	Programmable Switch Mode Cntrl	8-/12-/16-/20-/24-bit Timers	Peripheral Pin Select	ETHERNET MAC
Operational Amplifiers*	10b/12b/16b PWM	Quadrature Decoder	eXtreme Low Power XLP Technology	I ² C/TWI
Ramp Generator*	Waveform Extension	Output Compare	picoPower	LIN
Slope Compensation*	Clock Failure Detection	mTouch® solution	EEPROM	SPI™
Voltage Reference	Cyclical Redundancy Check	Qtouch Solution	External Bus Interface	Keeloq® Sub-GHz RF
Zero Cross Detect*	Hardware Limit Timer	Peripheral Touch Controller	Hardware Multiply	Crystal Free USB
High Current I/O*	Windowed WDT	LCD	Math Accelerator	Full-Speed USB Device w/w/o OTG
TEMP Indicator/Sensor	Brown-Out Detection			IRCOM

Figure 2: Core Independent Peripherals address various 8-bit MCU design areas.

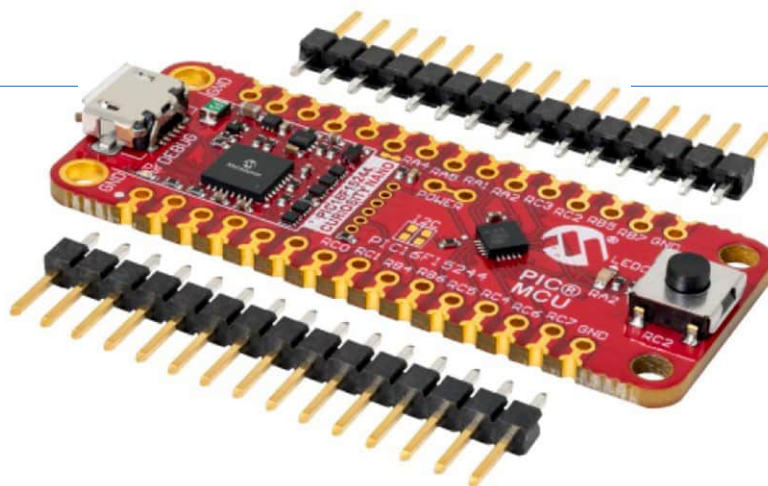


Figure 3:
The PIC16F15244
Curiosity Nano
evaluation board
and the two 100 mm,
1x15-pin header strips
in the Curiosity Nano
Evaluation Kit
simplify design.

peripheral category. The eight categories and their subcategories address most of the functionality expected in a cost-effective, embedded controller. Note that the green items provide additional power reduction possibilities to those mentioned earlier.

CIPs provide increased reliability by reducing the amount of code overhead. Functions implemented with hardware structures avoid potential software conflicts. In addition, peripheral interconnectivity in hardware reduces external connections, increasing the end system's reliability. The increased reliability of components decreases cost over the lifetime of the project.

The latest 8-bit MCU families offer extensive options in memory and pin counts, allowing developers to start with larger devices and scale down to smaller ones as the code is optimized. For example, the PIC16F152XX family is designed for cost-sensitive sensor and real-time control applications, featuring a 10-bit analog-to-digital Converter (ADC), Peripheral Pin Select (PPS), digital communication peripherals, and timers. Memory features include the Memory Access Partition (MAP) to support users in data protection and bootloader applications.

Design Tools to Accelerate and Simplify Applications

Advances in design tools have streamlined the development process, reducing the need for extensive coding. Tools such as MPLAB Code Configurator [2] (MCC) enable the generation of compact, efficient code, significantly shortening development times.

The PIC16F15244 Curiosity Nano Evaluation Kit [3] (Part Number: EV09Z19A) exemplifies this, offering comprehensive support for new designs with full programming and debugging capabilities (**Figure 3**).

MPLAB X IDE (integrated development environment) provides a versatile platform for developing code for 8-, 16- and 32-bit MCUs, facilitating hardware simulation and integration using third-party tools.

A Promising (and Cost-Effective) Future

Microcontrollers have evolved remarkably, with 8-bit MCUs leading the charge in memory, power consumption, packaging, and peripherals. They support complex applications with increased memory and simplified development processes, reducing both development and production costs. The adaptability and efficiency of today's 8-bit MCUs make them a preferred choice for numerous IoT applications, ensuring a bright and cost-effective future for embedded systems.

Today's 8-bit MCUs extend far beyond mere data collection; they are capable of collecting, processing and transmitting data across various IoT applications. The latest 8-bit models address the increasing complexity of these applications by offering significantly larger memory capacities and enhanced peripherals. However, for compact and cost-effective designs, such as sensors and basic real-time control applications, the streamlined features of the 8-bit PIC16F152xx family are particularly advantageous. Equipped with Core Independent Peripherals, these MCUs are a preferred choice for many designers. ◀

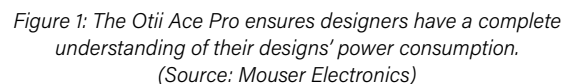
240306-01

WEB LINKS

- [1] nanoWatt XLP eXtreme Low Power PIC MCUs: <https://ww1.microchip.com/downloads/en/devicedoc/39941d.pdf>
- [2] MPLAB Code Configurator: <https://microchip.com/en-us/tools-resources/configure/mplab-code-configurator>
- [3] PIC16F15244 Curiosity Nano Evaluation Kit: <https://microchip.com/en-us/development-tool/EV09Z19A>

Advances Lead to More Efficient Use of Energy in Many Applications

The search for greater efficiency is a theme central to the electronics industry. Whether this is delivered by new and innovative solutions that use less energy or consume fewer raw materials, or through enhanced product development and maintenance, an increase in efficiency will often contribute to more sustainable designs.



Renewable energy generation is one of the most challenging areas for semiconductors, as sustainable goals demand that every bit of energy is converted to electrical power. Another area is the electric vehicle market, which requires as much range as possible to be extracted from a fixed-capacity battery.

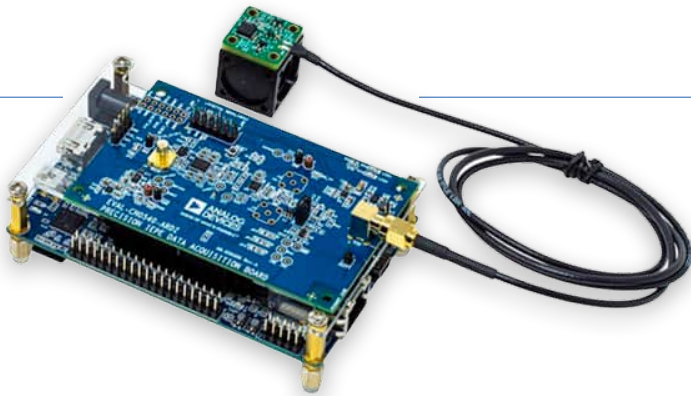


Figure 2: The Analog Devices CN0549 development platform monitors machines to help staff ensure that maintenance is performed only when necessary. (Source: Mouser Electronics)

The world has relied on silicon as the foundation of semiconductor devices for many decades. While it remains the preferred material in most applications, the most challenging uses demand efficiency levels that traditional silicon struggles to provide. Here, designers are looking to other materials, such as silicon carbide (SiC), that provide high reliability with ultra-low losses, even in high-frequency operation.

Take inverters, for example. Inverters are essential for converting DC from solar photovoltaic panels into AC grid power. Silicon can achieve 98% efficiency, which is good. However, moving to a SiC solution will halve the losses (99% efficiency). If SiC inverters were deployed across the US, where 60 GW of energy generation is solar, the savings would be 600 MW. In Europe, where over 200 GW of power is solar based, the savings would exceed 2 GW.

SiC devices may not yet be commonplace, but many vendors now offer these high-performance devices. Onsemi [4] offers a broad range of SiC products, including the M3S EliteSiC MOSFETs [5], which offer 40% lower total switching losses compared to earlier generations.


Sustainability in Industrial Maintenance

Factories are essential to producing the goods we consume in our lives, and will continue to produce sustainable goods in the future. In the past, they have been significant consumers of energy, but rising energy costs and the need for sustainability have led to a greater focus on saving energy.

For example, ultra-low-power sensors within factories and warehouses are often used to control lighting so that it operates only when humans are present, thereby eliminating waste.

Maintenance can be a tricky subject: If done too often, it wastes resources, while delaying it too long risks a breakdown and an expensive repair bill. Devices such as Analog Devices Inc's [6] CN0549 [7] condition-based monitoring development platform (**Figure 2**) — featuring the CN0540 IEPE Data Acquisition Board [8], CN-0532 Circuit Evaluation Board [9], and EVAL-XLMOUNT1 Mounting Block [10] — “listen” for machine vibrations that indicate wear and if maintenance is needed.

Conclusion

Sustainability will be a key theme for the future, and the technology industry will play a significant role. It will lead the way in generating electrical power even more efficiently as well as ensuring the efficient use of such a valuable resource through good design practices, automation, and monitoring. 



240281-01

About the Author

As Mouser Electronics' Director of Technical Content for EMEA, Mark Patrick is responsible for creating and circulating technical content within the region — content that is key to Mouser's strategy to support, inform, and inspire its engineering audience. Before leading Technical Content, Mark was part of Mouser's EMEA Supplier Marketing team and played a vital role in establishing and developing relationships with key manufacturing partners. Mark's previous experience encompasses hands-on engineering roles, technical support, semiconductor technical sales, and various marketing positions. A “hands-on” engineer at heart, Mark holds a first-class Honors Degree in Electronics Engineering from Coventry University. He is passionate about vintage synthesizers and British motorcycles, and thinks nothing of servicing or repairing either.

WEB LINKS

- [1] United Nations' Sustainable Development Goals (SDGs): <https://sdgs.un.org/goals>
- [2] IEA, “Energy Efficiency Policy Opportunities for Electric Motor-Driven Systems,” 2011: <https://iea.org/reports/energy-efficiency-policy-opportunities-for-electric-motor-driven-systems>
- [3] Qoitech Otii Ace Pro Power Supply and Measuring Instrument: <https://tinyurl.com/qoitech-otii>
- [4] Onsemi at Mouser: <https://tinyurl.com/manufacture-onsemi>
- [5] M3S EliteSiC MOSFETs: <https://tinyurl.com/M3S-EliteSiC>
- [6] Analog Devices at Mouser: <https://tinyurl.com/manufacture-AD>
- [7] CN0549 Dev Platform: <https://tinyurl.com/ADCN0549>
- [8] EVAL-CN0540-ARDZ: <https://tinyurl.com/ADCN0540>
- [9] EVAL-CN0532-EBZ: <https://tinyurl.com/ADCN-0532>
- [10] EVAL-XLMOUNT1: <https://tinyurl.com/EVAL-XLMOUNT1>

AWS for Arduino and Co. (1)

Using AWS IoT ExpressLink in Real Life

By Tam Hanna (Hungary)

Connecting to Amazon Web Services (AWS) can be a bit challenging. While it's still possible with a Raspberry Pi, single-board computers are often too large for many practical IoT applications. This is where AWS IoT ExpressLink helps, providing a simple way for small devices to connect to AWS. The AWS IoT ExpressLink powered connectivity module manages cloud communications, freeing up the host microcontroller to focus on measurements and control tasks. In this first article of a two-part series, we will walk through setting up an ExpressLink module using an ESP32.

Cloud services such as Microsoft Azure and Amazon Web Services (AWS) offer powerful options for storing, processing, and displaying data for their users. Communicating with cloud services isn't exactly simple, but it's doable for single-board computers such as a Raspberry Pi or Orange Pi. These systems, however, are often too big for many practical use cases. When it comes to microcontroller boards, implementing a fully-fledged TCP/IP stack on a 32-bit controller is possible, but is resource-sapping. On smaller, less capable machines, this task becomes even more challenging.

With AWS IoT ExpressLink technology, Amazon aims to help designers of small hardware devices to overcome this tricky last mile. In this article, we'll explore the possibilities and limitations of the system, followed by experiments using an Arduino UNO R4 WiFi and an Espressif ESP32-C3-AWS-ExpressLink-DevKit. The goal is to present an "end-to-end workflow" that you can use productively in your own projects.

The AT Protocol

The AT protocol, originally developed by Hayes Microcomputer Co. to control a modem, did not die along with dial-up. In fact, the control protocol, in its various guises, is alive and well, as anyone who has ever dabbled with the control of modern radio modules will tell you.

The best summary of how an ExpressLink module works can be found in Espressif's documentation. **Figure 1** shows how the individual elements work together. The ExpressLink module communicates with the host microcontroller using a simple

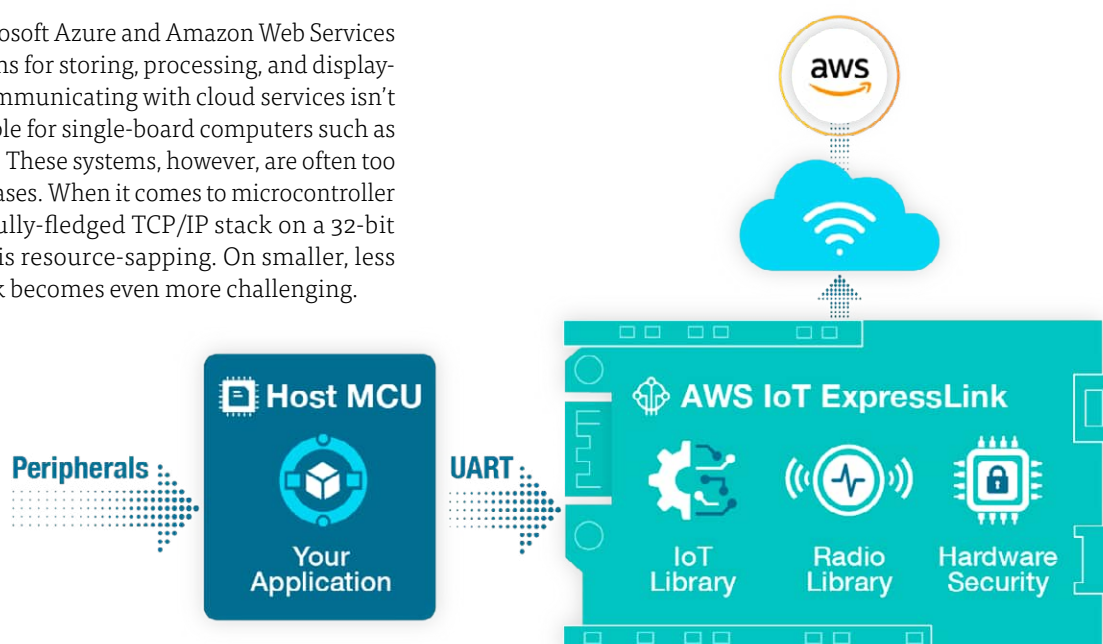


Figure 1: Some users refer to the ExpressLink module as the "AWS modem."
(Source: [1])

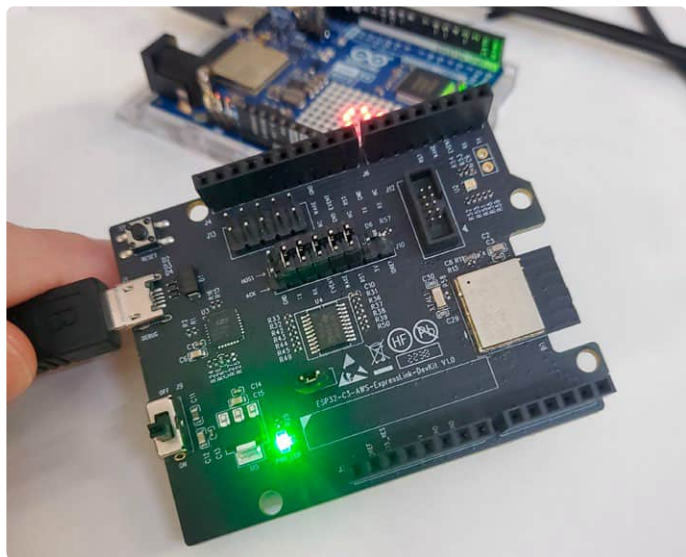


Figure 2: This module brings success!

interface, which is solely responsible for interacting with sensors and actuators. The host microcontroller software has the task of sending correctly formatted commands to the module, which handles communication with the cloud.

It's worth noting here that the ExpressLink module will typically also handle the air interface. Those who wish to combine such a module with a fully-fledged radio SoC, such as the GigaDevice GD32W515 or another ESP32 board, must ensure that cloud communication occurs through the ExpressLink module transceiver.

One benefit of this design decision, however, is that most ExpressLink modules also come with an antenna — we don't need to bother about the antenna design or vector network analysis etc.

The ESP32-C3-AWS-ExpressLink-DevKit shown in **Figure 2** is a neat starter kit that can be paired with various Arduino controller boards. According to oemsecrets.com, its best price is around 28 euros (see [2]), it's an accessible way to get started with AWS IoT ExpressLink. In the following steps, we will use an Arduino UNO R4 WiFi as the baseboard, so its Wi-Fi module will be unused (see above).

Setting Up the Hardware

In theory, getting the board up and running online couldn't be easier — Espressif offers the ExpressLink module in the form of an Arduino shield, so all we need to do is push the two boards together.

Figure 1 is, however, a bit misleading here because the communication interface between the host MCU and the radio module isn't just a simple serial link. **Figure 3** shows some additional connections that also need to be routed in order to relay the operational state of some relevant control signals.

Since we're focusing on the combination of Arduino and the Espressif Arduino shield in the following steps, we won't dwell on these pins here. They're mentioned mainly because Espressif emphasizes their importance in the documentation — if they're missing, the design will only be able to handle a "Hello World" communication and any practical communication with AWS will not be possible.

ExpressLink Pin	ESP32-C3 GPIO Pin	ESP32-C3-MINI-1-N4-A Module Pin
TX	IO19	27
RX	IO18	26
EVENT	IO10	16
WAKE	IO3	6
RESET	EN	8

Figure 3: Some GPIO pins are required for signaling critical or otherwise relevant operational states. (Source: [14])

Before plugging the Arduino and Espressif module card together, it's worthwhile performing an update of the module's firmware. You cannot be sure how long the module has been sitting on the distributor's shelf, and won't know whether the latest version of software is preloaded.

In theory, we could do this update over the air interface, but this would require an already-existing connection between the module and the AWS IoT cloud services.

As an alternative, Espressif has provided a Python script that requires a version of Python 3 running on a PC. In my case, a workstation running Ubuntu 20.04 LTS, shows the version status as:

```
tamhan@TAMHAN18:~$ python3 --version
Python 3.8.10
```

For actual communication, you'll need to enter the command `pip3 install pyserial==3.5`, which provides a bridge library between the Python runtime and the PC serial ports.

Next, you'll need to download the tool, which is available as a .py file, for firmware downloading. You can find it on GitHub [3]. Look for the button highlighted in **Figure 4**, which allows direct downloading of the tool without the usual copy-paste process.

After this, you'll need to download the firmware — for this you need to visit GitHub again [4]. We managed to download the firmware version `v2.5.0.bin` — just make sure you do not grab the hash file by mistake.

Updating the operating software cannot be done via the USB interface built into the board — it connects to an "application UART" and doesn't allow programming of the ESP32 what serves here as the radio module. As a workaround, we used an FTDI mini-module,



Figure 4: This button saves valuable time.

```

[ 1613.074190] usb 1-1.1: new high-speed USB device number 7 using ehci-pci
[ 1613.191003] usb 1-1.1: New USB device found, idVendor=0403, idProduct=6010, bcdDevice= 7.00
[ 1613.191006] usb 1-1.1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 1613.191007] usb 1-1.1: Product: FT2232H-56Q MiniModule
[ 1613.191009] usb 1-1.1: Manufacturer: FTDI
[ 1613.191010] usb 1-1.1: SerialNumber: FTL6Z5PC
[ 1613.235698] usbcore: registered new interface driver ftdi_sio
[ 1613.235776] usbserial: USB Serial support registered for FTDI USB Serial Device
[ 1613.235898] ftdi_sio 1-1.1:1.0: FTDI USB Serial Device converter detected
[ 1613.236007] usb 1-1.1: Detected FT2232H
[ 1613.237935] usb 1-1.1: FTDI USB Serial Device converter now attached to ttyUSB0
[ 1613.238114] ftdi_sio 1-1.1:1.1: FTDI USB Serial Device converter detected
[ 1613.238940] usb 1-1.1: Detected FT2232H
[ 1613.239673] usb 1-1.1: FTDI USB Serial Device converter now attached to ttyUSB1
[ 1676.337202] usb 1-1.4: new full-speed USB device number 8 using ehci-pci
[ 1676.452057] usb 1-1.4: New USB device found, idVendor=10c4, idProduct=ea60, bcdDevice= 1.00
[ 1676.452062] usb 1-1.4: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 1676.452065] usb 1-1.4: Product: CP2102N USB to UART Bridge Controller
[ 1676.452067] usb 1-1.4: Manufacturer: Silicon Labs
[ 1676.452069] usb 1-1.4: SerialNumber: 5a66658d521fed11a7015cb1dcc2a201
[ 1676.452682] cp210x 1-1.4:1.0: cp210x converter detected
[ 1676.455540] usb 1-1.4: cp210x converter now attached to ttyUSB2
tamhan@TAMHAN18:~$

```

Figure 5: Both USB-to-serial converters appear in the workstation /dev file system.

```

tamhan@TAMHAN18:~/Desktop/Stuff/2024Mar/EAG-ExpressLink$ python3 otw.py /dev/ttyUSB1 v2.5.0.bin
File size 1511424
Uploaded 100.0%
Done...
tamhan@TAMHAN18:~/Desktop/Stuff/2024Mar/EAG-ExpressLink$

```

Figure 6: Ensure that the module is up-to-date. Otherwise, OTW.py displays error messages.

specifically the FT2232H MINI MODULE, which also works well for programming ESP32 microcontrollers in the lab.

It's worth noting that the serial connection and pin assignment is non-standard. Pin BD1 is connected to the TX of the ESP32 module, while Pin BDO is connected to the RX pin. The TX and RX pins are in the middle of the module and do not comply with the usual Arduino pin layout.

Finally, we need a jumper wire to connect the two ground potentials together. For the actual setup, the FTDI module is first connected to the computer, and the power supply of the ExpressLink board is then connected afterwards. This connection setup is sequenced in *dmesg* as shown in **Figure 5**.

The key point here is to note that ports ttyUSB0 and ttyUSB1 are responsible for PC-FTDI communication. If you use the pins mentioned, the program transfer runs according to the following sequence:

```

tamhan@TAMHAN18:~/Desktop/Stuff/2024Mar/EAG-ExpressLink$
python3 otw.py /dev/ttyUSB1 v2.5.0.bin

```

Success is evident, as shown in **Figure 6**. After shutting down, we won't need to use the FTDI module again. The ExpressLink module will now only be connected to the Arduino.

Setting up the Cloud Connection

After updating the firmware on the module, we're now ready to put it into operation. In the following steps, I assume you will already have your own ready-configured AWS account and can log in as a

root user at <https://aws.amazon.com>. For practical security reasons, we'll skip the recommended use of IAM in the following steps due to space constraints.

If you don't have an AWS account yet, check out the quick-start guide at [5]. Please note that to set this up requires a credit card.

The module with its fixed firmware is initialized via the cloud or by sending AT commands with specific configuration parameters. For the following steps, you'll need the AWS IoT console, available at <http://console.aws.amazon.com/iot>, running on your workstation, which you can keep open in a browser window.

In the following steps, Arduino IDE 2 will be used as the development environment for the Arduino sketch — we will not be going deeper into its deployment with Linux using an AppImage file.

It's important to note that during this operation, only the Arduino needs to be connected to the computer — the module is powered from the connected Arduino. If the PC needs to communicate directly with the module by sending AT commands, the connection to the Arduino must first be disconnected, and a Micro-USB cable used with the module.

The hardware communicating with AWS is represented on the server as a *Thing*. For the authentication of our *Thing* with the server, both a *Thing Name* and a cryptographic certificate are required. Both are located in the ExpressLink module memory and must now be ascertained.

Although Amazon provides the `AT+CONF?` *ThingName* and `AT+CONF?`


```
I (784) wifi_init: tcp mss: 1440
I (784) wifi_init: WiFi IRAM OP enabled
I (794) wifi_init: WiFi RX IRAM OP enabled
I (804) phy_init: phy_version 970,1856f88,May 10 2023,17:44:12
I (844) wifi:mode : sta (dc:54:75:90:4e:44)
I (844) wifi:enable tsf
I (854) PF_WIFI: Station started event
I (854) PF_TIME: Initializing the NTP
I (854) esp_secure_cert: Partition found, current format is cust flash legacy.
I (864) esp_secure_cert: Current partition format: CUST_FLASH_LEGACY, partition name: pre_prov
I (874) esp_pre_init: Device Certificate:
Length: 1318
-----BEGIN CERTIFICATE-----
MIIDnzCCAoegAwIBAgIRANQdVFip+kQjsu6eqj1obXswDQYJKoZIhvcNAQELBQAw
H1ELMAkGA1UEBhMC004xczAxbGlnZy5kaWVudC5jb20wDQYJKoZIhvcNAQEBBQAw
R0EwDQYJKoZIhvcNAQEBBQADggGPADCCAQAwgBgwggEoMBIGA1UdEQQOMAsGA1Ud
EwEPAAQDAgAwgBgwggEoMBIGA1UdEQQOMAsGA1UdEwEPAAQDAgAwgBgwggEoMBIG
A1UdEQQOMAsGA1UdEwEPAAQDAgAwgBgwggEoMBIGA1UdEQQOMAsGA1UdEwEPAAQDAg
Edc+ngHHLlNq...SP2NDNSIK3EKRDYT9YCJO/KaSVB
/iSE4SLsp9/HI...HJ)
-----END CERTIFICATE-----

I (994) esp_pre_init: Thingname is: 05cdbf4f-ab1f-4bdf-96...
I (1004) EL: Starting ExpressLink
I (1004) PF_KVS: Get Failed DefenderPeriod
I (1014) ELEVENT: Setting Event pin high.
I (1014) PF_MAIN: UART task started, input is now processed
I (1014) main task: Returned from app main()
```

`Certificate pem` commands in its documentation, in practice, it's easier to connect the module to the computer directly, start a terminal program, and then press the reset button. The firmware implemented by Espressif provides the required information in this case — as shown in **Figure 7** — which can then be copied to the clipboard directly.

```
tamhan@TAMHAN18:~$ screen /dev/ttyUSB0 115200
```

```
tamhan@TAMHAN18:~$ sudo killall screen -s9
```

Navigation is a little tricky here, as the left-hand navigation interface is both incomplete and scrollable. Nonetheless, the next action required is to click on the *Link Manage* button — the required option isn't available in the tree. Instead, you'll need to click on the *Create Thing* button on the screen that loads next, as shown in **Figure 8**.

now guide us through the three steps to incorporate a new *Thing* into the AWS IoT service.

Here, in the first step, we opt for the *Use my certificate* option followed by selecting *CA is not registered with AWS IoT*.

```
tamhan@TAMHAN18:~/Desktop/ExpressLink$ touch tamscert.pem
tamhan@TAMHAN18:~/Desktop/ExpressLink$ gedit tamscert.pem
```

Figure 8: Only this option allows the creation of new Things.

Starter Boards

In this article, we have used the Espressif module because of its availability. Amazon has signed contracts with various other partners, such as U-blox, Infineon, Realtek and Telit — you can find an up-to-date list of starter boards from all manufacturers under [15].

After entering the `gedit` command, a new file appears in the text editor, where we paste the certificate text we copied earlier. It's important to include the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` delimiters in the text file, or the parser won't be able to retrieve the certificate information.

Once you've saved the `.pem` file, you can upload it and in the last step. In the *Certificate status* section, select the *Active* option. Then, click on *Next Step* to proceed to the third step, then *Attach policies to certificate*.

AWS IoT implements a certificate-based authorization scheme. This means that certificates are associated with policy documents, which then dictate which actions the respective resource in the AWS network is allowed to perform.

In newly configured AWS clusters, predefined policies haven't been used for some time now. Amazon likely adopts this approach to avoid liability in cases of security incidents caused by insecure configurations.

For this reason, in the first step, we can click on the *Create Policy* button, which opens a new browser window showing the Policy Editor. In the first step there, we give the policy a name by which it will be known henceforth.

Below that, there's a second option responsible for setting the actual permissions granted by the policy. Here, we click on the *JSON* option to enable direct text input. The code already there is generally ready for use — just make sure to insert the wildcard placeholders shown in the following scheme into the *Action* and *Resource* fields:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}
```

The policy shown allows the device universal access to everything in the AWS backend. From a logical standpoint, this approach isn't advisable in practical deployments. It's worth noting again

for legal completeness that you should eliminate these policies as soon as possible after experimentation. Both the publisher and author cannot be held liable for the consequences of any mistakes made.

After clicking the *Create* option, the Policy Editor window opens a list of policies — it's only necessary now to close the window and return to the actual Thing generation process. The newly created policy is now ready for selection.

Next, it's necessary to click the *Create* button to complete the entry process at the backend.

To finish the configuration, you'll need to scroll all the way down in the left-hand navigation interface of the AWS IoT console and click on the *Settings* option.

In the *Device Data Endpoints* section, you'll find a string with the following example format: `a3gw111abcdefghdl-ats.iot.eu-west-1.amazonaws.com`. This needs to be saved — it determines through which entry point the module will connect to the cloud.

Establishing the Wi-Fi Link

Correct configuration on the AWS cloud side is only half the battle, as the ESP32 module still needs to be able to communicate with your local network. To achieve this, you need to write your Wi-Fi credentials into the Secure Element.

In the AT command set specified by Amazon, you'll find the commands `AT+CONF SSID=<replace-with-your-router-ssid>` and `AT+CONF Passphrase=<replace-with-your-router-passphrase>`, which allow us to transfer the appropriate Wi-Fi credentials via a serial link.

Espressif has, however, equipped its ExpressLink modules with a useful and convenient feature: Android and iOS apps, which allow for a more "graphical" configuration of the modules, can be found at [6] and [7].

The only problem in this context is that exposing the necessary Bluetooth interface first requires transmitting the command `AT+CONFMODE`. For this, we can use the example sketch provided by Amazon [8].

Normally, the Arduino uses the hardware serial port for outputting `printf()` messages and other status information. After attaching an ExpressLink module, this port (logically) isn't available.

The AWS development team has instead relied on the *SoftwareSerial* library. At the time of writing, however, this library doesn't work particularly well with the Arduino R4 (see [10]), but it isn't required anyway. Our first job is thus to clean up the sketch by removing all related calls. It's very useful that the Renesas Arduino offers two hardware UART interfaces using different pins [11]. *Serial1* sends data to the radio module, while *Serial* can be used for communication via Arduino IDE's Serial Monitor running on the PC. This means that the method responsible for communication with the ESP32 module needs to be modified according to the following commands:

```
String execute_command(String command,
    unsigned long timeout_ms) {
    Serial.print("EXC : ");
    Serial.println(command);

    unsigned long saved_timeout_value_ms =
        Serial1.getTimeout();
    Serial1.setTimeout(timeout_ms);
    Serial1.println(command);
    String s = Serial1.readStringUntil('\n');
    s.trim();
    Serial1.setTimeout(saved_timeout_value_ms);
    return s;
}
```

Be sure to change the configuration according to the following definitions before executing the program:

```
#define EVENT_PIN 2
#define RESET_PIN 4

#define MY_AWS_IOT_ENDPOINT "change_me-ats.iot.eu-west-1.
    amazonaws.com"
```

Due to a conflict reported by the author between the Renesas and Arduino parts of the library, as documented under [12], an adjustment of the *process_event()* function is required. Specifically, the value returned by *c_str()* must not be passed directly to *scanf()*:

```
event_t process_event()
{
    String response;

    int val = 0;
    event_t event_number = EVENT_NONE;
    val = digitalRead(EVENT_PIN);
    if(val)
    {
        response = execute_command("AT+EVENT?",
            3000);
        if (response.equals("OK"))
        {
            return EVENT_NONE;
        }
    }
}
```

```
    }
    else {
        char ok_string[3];
        int topic_index;
        int total_read;
        char someResponse[300];
        strcpy(someResponse, response.c_str());
        total_read = sscanf(someResponse,
            "%s %d %d %s", ok_string,
            &event_number, &topic_index);
        return event_number;
    }
}
```

The ESP32 module is connected to the Arduino, while only the Arduino uses the USB cable link. The power supply on the ExpressLink Kit should be turned off. A rapid double-press of the Arduino UNO R4's RST-button is required to enable the bootloader to accept new compilations.

After successfully adjusting and flashing the test sketch [13], the module receives the necessary command if it's not already in a provisioned state. By the way, the verification of this can be done quite neatly by attempting to read back the SSID stored in the Secure Element:

```
void loop() {
    event_t event = EVENT_NONE;
    String response;
    static state_t state = STATE_INIT;
    if (state != STATE_INIT) {
        event = process_event();
    }
    switch (state) {
        ...
        case STATE_EL_READY:
            response = execute_command(
                "AT+CONF Endpoint="MY_AWS_IOT_ENDPOINT"",
                3000);
            response = execute_command(
                "AT+CONF Topic1=TEST", 3000);
            response = execute_command(
                "AT+CONF? SSID", 3000);
            state = process_ssid(response);
            break;
        case STATE_UNPROVISIONED:
            response = execute_command(
                "AT+CONFMODE", 5000);
            if (response.equals("OK CONFMODE ENABLED"))
            {
                state = STATE_PROVISIONING;
            }
            break;
    }
}
```

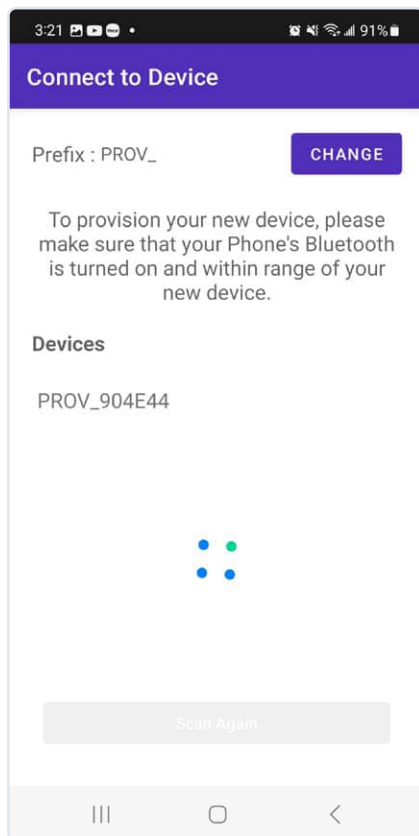



Figure 9: The Bluetooth LE scanner has found a new victim!

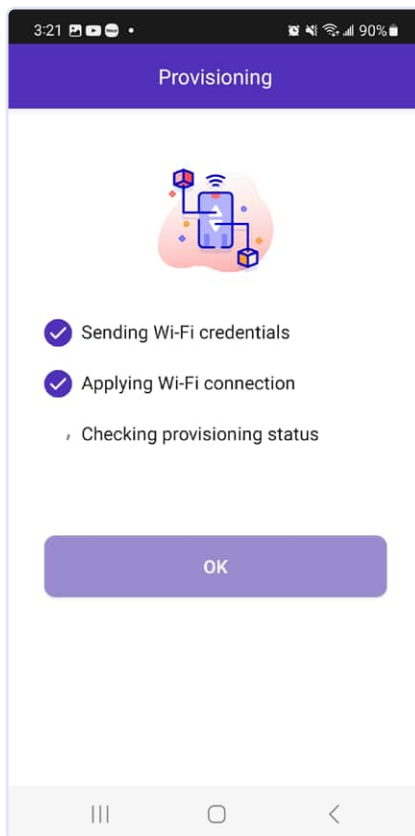


Figure 10: Provisioning can take up to 2 minutes...

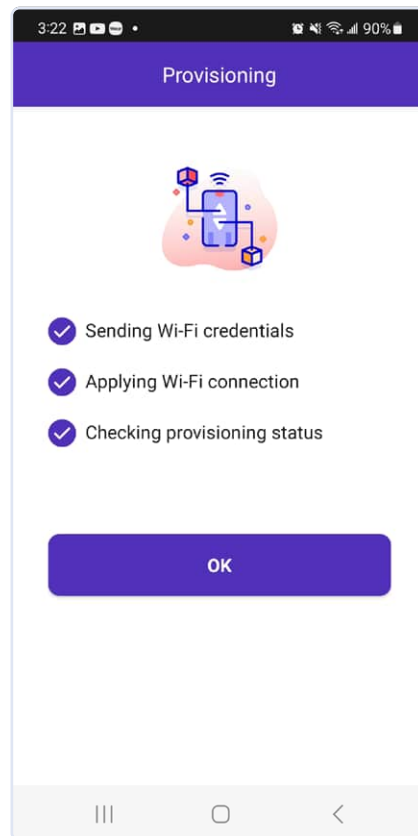


Figure 11: ...and is only complete when the window shown here appears.

Amazon implements the program shown here as a colossal state machine. The method responsible for processing the SSID then sets the correct state according to the following scheme should the returned SSID be unusable:

```
state_t process_ssid(String response)
{
    int event_number = 0;
    char ok_string[3];
    char ssid_string[33] = {0};
    int total_read;

    total_read = sscanf(response.c_str(),
        "%s %s" , ok_string, ssid_string);

    if(total_read > 1) {
        return STATE_PROVISIONED;
    }
    else {
        return STATE_UNPROVISIONED;
    }
}
```

Once the module achieves Provisioning Status, it becomes visible to a Bluetooth LE scanner. A more convenient method, however, is

to use the smartphone app mentioned earlier. Due to peculiarities in the Android permissions system, the operating system responds with a barrage of permission requests on first launch. If you grant these, you'll find yourself on a camera display — Espressif uses the same application not only for Bluetooth LE provisioning, but also for capturing ExpressLink modules that provide their module name via a QR code.

In **Figure 9**, you can see how the module is discovered. The actual program execution is then carried out by a wizard that prompts you to enter the password after selecting the Wi-Fi name. Note that the transition between screenshots shown in **Figures 10** and **11** can take a little time.

To Wrap Up

In this article, so far, we've only demonstrated the setup of an AWS IoT ExpressLink module. In the second part of this series, we'll delve deeper into data transmission. The benefits of this particular system setup should be clear even now. Using such minimal hardware investment, it would otherwise be quite challenging to connect to a major cloud provider. The extra cost for the radio module units can be justified because, in good cooperation with Amazon, you can typically receive a promotional boost occasionally, and this may be of interest. ◀

240240-01

About the Author

Tam Hanna is an engineer who has been working with electronics, computers and software for over 20 years. He is a freelance designer, author of books, and journalist — [instagram.com/tam.hanna](https://www.instagram.com/tam.hanna). In his free time, he tinkers with 3D printing and, among other things, trades and enjoys high-end cigars.

Questions or Comments?

Do you have any questions or comments about this article? Email the author at tamhan@tamoggemon.com or contact the Elektor team at editor@elektor.com.



Related Products

> **Arduino Uno R4 WiFi**
www.elektor.com/20528



WEB LINKS

- [1] ExpressLink documentation from Espressif: <https://espressif.com/en/solutions/device-connectivity/esp-aws-iot-expresslink>
- [2] ESP32-C3-AWS-ExpressLink-DevKit Distributors: <https://oemsecrets.com/compare/ESP32-C3-AWS-ExpressLink-DevKit>
- [3] Download tool: <https://github.com/espressif/esp-aws-expresslink-eval/blob/main/tools/otw.py>
- [4] Module firmware: <https://github.com/espressif/esp-aws-expresslink-eval/releases>
- [5] AWS getting started guide: <https://tinyurl.com/elgsg-set-up>
- [6] Description of the .pem file format: <https://tinyurl.com/PEM-file>
- [7] Android Config tool: <https://play.google.com/store/apps/details?id=com.espressif.provble>
- [8] iOS Config tool: <https://apps.apple.com/app/esp-ble-provisioning/id1473590141>
- [9] Arduino-Sketch for Bluetooth activation: <https://tinyurl.com/sketches-arduino>
- [10] Software UART issues: <https://github.com/arduino/uno-r4-library-compatibility/issues/12>
- [11] Two Hardware UARTs: <https://forum.arduino.cc/t/uno-r4-and-serial-rx-tx/1146022/2>
- [12] Library issues: <https://github.com/espressif/esp-aws-expresslink-eval/issues/23>
- [13] The author's sketch: <https://elektormagazine.com/240240-01>
- [14] Getting Started Guide ExpressLink Evaluation Kit: <https://github.com/espressif/esp-aws-expresslink-eval>
- [15] ExpressLink Starter Boards: <https://tinyurl.com/ExpressLink-Starter>

Join our Community



www.elektormagazine.com/community





Airflow Detector Using Arduino Only

No External Sensors Needed!

By Raymond Schouten (The Netherlands)

This detector circuit uses just the Arduino Nano itself, without an added sensor. It can detect air flows blown from a distance of 30 cm, by measuring the internal heated chip temperature with 0.01°C resolution and finding rapid changes. This design uses a dithering technique, enabling increased resolution beyond that of an ADC LSB.

What is behind the airflow detection concept in this design? It's quite a straightforward principle. When running, the Arduino Nano's ATmega328P controller heats up to some 6...10°C above the ambient temperature. An increase in airflow toward that chip will then cool it down: This is the basis of detecting a sudden air flow around the Arduino Nano board.

This detection is done by comparing the new temperature readout to the previous result twice per second. Apart from detecting airflow, it also detects other temperature changes, such as moving the board or touching the chip package (**Figure 1**).

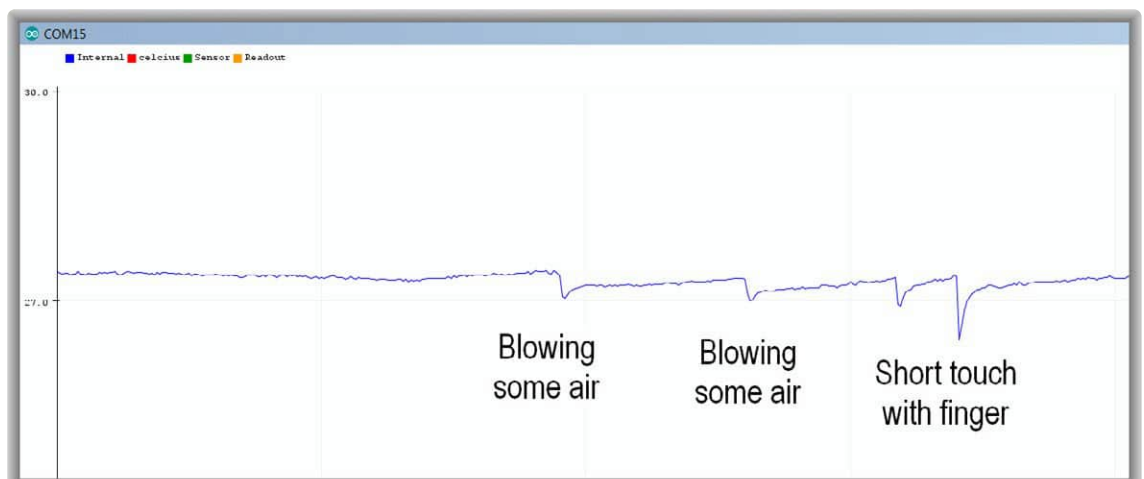
Please note that:

- To work as a detector, after cold boot, you need to give the Arduino some time to warm up (5...10 min). Once it has reached a "stable" temperature, it is possible to start detecting these small sudden air flows.
- This is a very sensitive detector — readout fluctuations can be <0.02°C — but an airflow causes larger fluctuations, readout stability evaluation can be done by shielding it from airflow, for example using a plastic bag. See **Figure 2**.

The Temperature Readout Concept

The chip's internal ADC can not only read its analog input pins, but can also be set to read the on-chip temperature sensor. This voltage will always be <1 V. To increase ADC sensitivity, the reference voltage is set (in software) to a 1.1 V internal source instead of the default 5 V supply. In a 10-bit ADC (1,024 steps), the smallest readable step then lowers from 5 mV to 1 mV (rounded figures).

Figure 1: The rapid response in temperature readings from the thermal sensor included on the Arduino Nano board's ATmega328 controller.



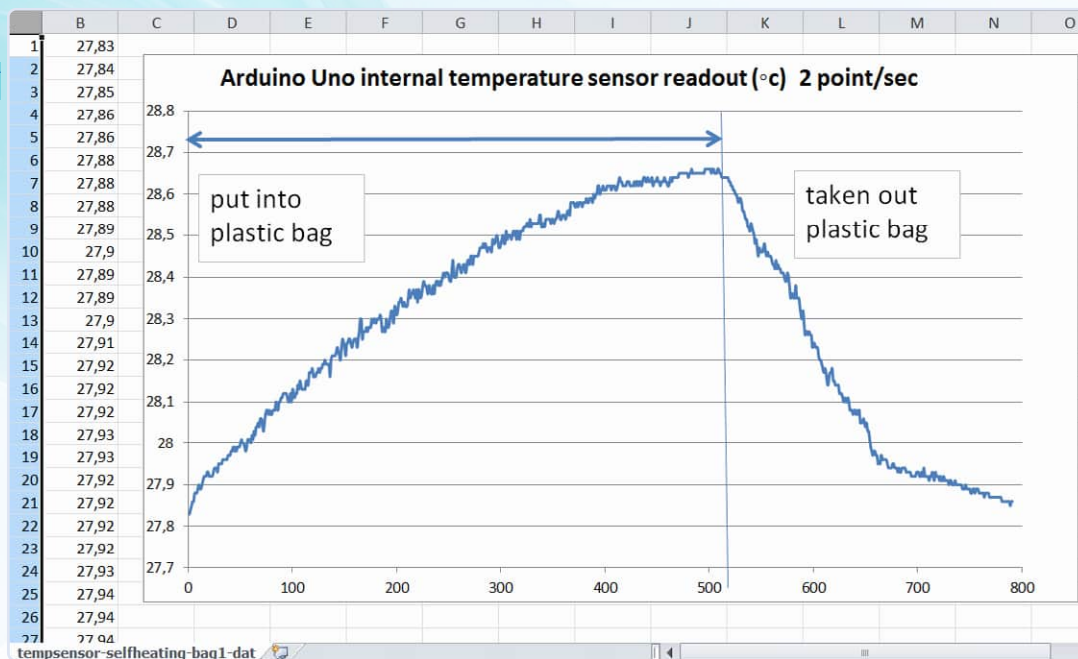


Figure 2: For evaluating the readout stability, it's advisable to have the board working in a plastic bag, to shield it from any unwanted airflow changes.

This smallest step is expressed through the LSB (least-significant bit). According to the datasheet, the output from the sensor is 1 mV/°C, so, with a 1 mV/step, we obtain an overall measurement resolution of 1°C — far too coarse for our purposes. The next block explains how this can be improved to approximately 0.01°C/step.

Increasing the ADC Readout Resolution by Dithering

As mentioned earlier, an ADC is usually limited to detecting changes (steps) that have the magnitude of an LSB. Having lowered the reference voltage to 1.1 V, we've got approximately 1 mV of magnitude for the LSB.

The ADC then gives a readout scaled in multiples of 1 mV. If, for example, the input is somewhere between 99.5 mV and 100.5 mV, it will always give "100 mV" readings. So if we have an input signal of 100.1 mV, we just read "100 mV". Here, the concept of dithering can help us out. This technique is one of the rare applications where the presence of noise improves things.

To better understand how dithering works, let's draw an example for our specific case, as shown in **Figure 3**. If we add random noise to the 100.1 mV signal (for now: let's assume noise with 1 LSB = 1 mV amplitude) and quickly do many repeated measurements, we will get around 90% "100 mV" readings and 10% "101 mV". We can calculate the average and conclude that we actually had a 100.1 mV input signal. For larger noise amplitudes, the averaged results also lead toward this value.

The fluctuations induced by the noise make this result not perfectly stable, but, as a rule of thumb, you get a 10× improvement for 100-sample averaging. The improvement is the square root of the number of samples. In this application, 6,400 samples are averaged for each temperature reading. The square root of that predicts an 80-fold improvement.

We started with a 1°C resolution, so we could theoretically improve to 0.0125°C resolution. The test data visible on the right side of **Figure 4**

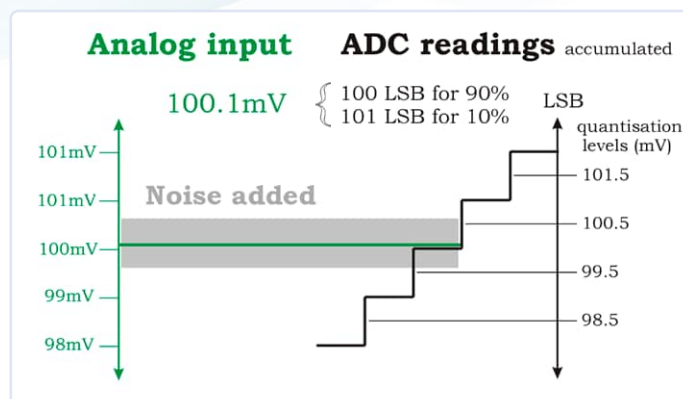


Figure 3: The readings of the noise component are added to the one related to the primary signal, then an averaged result is calculated from the multitude of these readings.

shows a trend of 0.01...0.02°C fluctuating steps in temperature readout, which indicates that we've come close in practice. See the data file downloadable at the Elektor Labs page for this project [1] and the results in the demonstration video [2].

Precision Vs. Accuracy

Note that the readout has high precision (fluctuations of <0.02°C) but is not at all accurate to that level, meaning that the low-fluctuation



Figure 4: The Arduino Nano board used in this project, with the flow of the readings showing on the right.

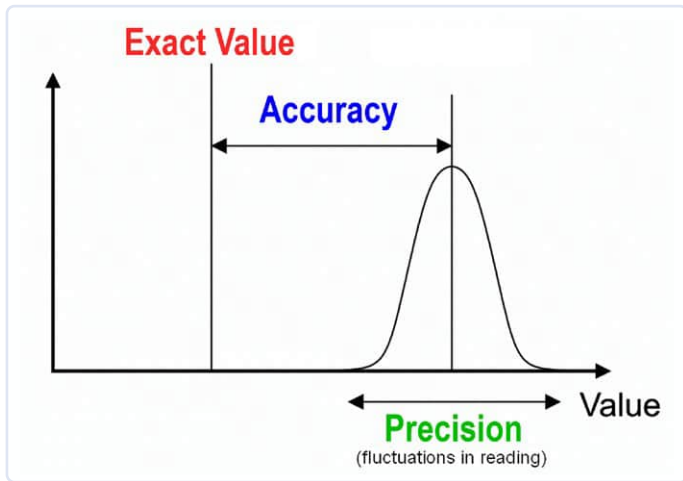


Figure 5: Relationship between precision and accuracy. (Source: Wikimedia Commons, Pekaje / Anthony Cutler, https://commons.wikimedia.org/wiki/File:Accuracy_and_precision.svg)

How did I come to this conclusion? I could still see most readings centering around the 5 mV LSB steps. I plan to show how this can be solved to make a high-resolution voltmeter in a future post.

Some Suggestions (Untested)

Turn It Into a Thermometer

If you lower the average dissipation of the controller, you could use this design as a thermometer. To get this functionality, put the controller in sleep mode and wake it once every few seconds to measure.

Read External Sensors

You could connect external diodes to the analog input pins, using a weak pull-up resistor to bias it with a small current. In this way, you could presumably read multiple external temperatures with high precision (and low accuracy) using the same code concept. ◀

220615-01

readings (precision) can be offset several degrees from the actual value (accuracy), as illustrated in **Figure 5**. For applications detecting small relative changes in the temperature, but not absolute readings, you only need high precision, so this works well here.

Software

Here, an Arduino Nano was used, but other boards with the same ATmega328P controller (such as Arduino UNO or Arduino Pro Mini) should also work. The source code for this project is also available at [1], and is rather straightforward in its phases:

1. Initialize by setting the ADC reference and input multiplexer appropriately for reading the internal temperature sensor.
2. Accumulate 6,400 ADC readings to determine the temperature with high precision.
3. Send the result via USB.
4. Check if the readout dropped suddenly. If so, switch off the indicator LED; if not, switch it on.
5. Go to phase 2.

Limitations

- This works best on Arduino boards equipped with the SMD version of the microcontroller. The larger DIL package resists temperature changes for longer, is more thermally isolated, and thus unsuitable for this application.
- The dithering process slows down the actual measurement rate, since we have to collect many readings. For this, we use a 10 kHz ADC, slowed down to about two readings per second. For temperature measurements, this is not an issue.
- An ADC is not perfect, so if its smallest steps are unequal, you get additional errors in the statistics. You might get some insight on DNL (differential non-linearity) in [3].
- For the Pros: Larger than 1 LSB amplitude noise also helps to average out the ADC DNL errors somewhat.
- If the noise does not have an equal opportunity of distribution between all values, then the statistics are also affected.

How Did We Add the Noise Source Needed For The Dithering?

Well, we didn't. At these small 1 mV LSB levels we get it for free here from the circuit elements, such as the sensor itself. As a comparison: When reading an input voltage using the 5 V ADC reference, I found that "natural" ambient noise was too low to make the dithering work properly.



About the Author

Aside from his daytime job developing low-noise instrumentation electronics, Raymond Schouten runs hobby projects designing tiny music synthesizers and other compact circuits. Most of his designs aim to achieve maximum results with the simplest hardware. He is also frequent in posting projects on Elektor Labs, YouTube, and on his personal website at rs-elc.nl.

Questions or Comments?

Do you have technical questions or comments about this article? Feel free to write to the author at rs.elc.projects@gmail.com or to the Elektor Editorial Team at editor@elektor.com.



Related Products

- **Arduino Nano**
www.elektor.com/17002
- **Ashwin Pajankar, Kickstart to Arduino Nano (Elektor, 2022)**
Book: www.elektor.com/20241
E-Book: www.elektor.com/20242

WEB LINKS

- [1] Elektor Labs page for this article:
<https://tinyurl.com/airflowdetector>
- [2] Demo-movie on YouTube: <https://youtu.be/0p6hssAf7Xs>
- [3] Short wiki-note about DNL :
https://en.wikipedia.org/wiki/Differential_nonlinearity

Water Leak Detector

Source: Adobe Stock

Connected to Arduino Cloud



Figure 1: My water consumption meter with flow sensor attached.

By Yves Bourdon (France)

When a water pipe under my house cracked without me noticing it, it caused me several thousand Euros of damage. Therefore, I decided to create and install a water-consumption surveillance system to prevent such costly accidents from going unnoticed in the future. Using Arduino Cloud, I can now monitor my water consumption on my mobile phone any time of the day from anywhere in the world. Here is how I achieved this.

Water is precious and is becoming more and more expensive. A year ago, a water pipe cracked under my house without me noticing it immediately, and it cost me several thousand euros in water! So, I looked for a way to be warned in case if it happened again. The water department installed a meter incorporating a flow sensor (**Figure 1**) and a small wireless remote display. Unfortunately, reliability was not up to par, and there was no way of alerting me apart from an 'F' on the display supposed to indicate a leak...

Following the Elektor article presenting the water leak detector developed by Denis Lafourcade [1], I decided to take up his project (with his kind permission). I modified it quite extensively though to integrate it into my existing system. Also, I wanted to use Arduino Cloud, as presented in the 2022 Elektor Arduino Guest Edition [2].

Prerequisites

I wanted to be able to access my data on my mobile phone, and the Arduino Cloud environment is particularly suitable for this purpose. This allowed me to have a dashboard on my phone without having to develop a specific application that needs constant updating. Security being important, I especially did not want to open a port on my router to access my data. And, finally, in case of a leak, I wanted to be alerted by a push notification together with an audible alarm.

Specifications

- > Usage of Arduino Cloud to collect data and display statistics. A graph shows daily water consumption in real-time. Up to three months of data are stored in the cloud. Additionally, it's easy to download a CSV (comma-separated values) file for data processing in, e.g., a spreadsheet.
- > Usage of an ESP32 processor, and the Heltec WiFi Kit 32 development board [3] specifically, which includes, among other things, an OLED screen and a battery charger.
- > Direct connection (via an optocoupler to eliminate interference) to the reed sensor integrated into my water meter, located

approximately 40 meters away, against the wall of my property. Note that if you don't have a meter equipped with a sensor, you can easily find reed relay adapters for any existing meter (Figure 2).

- By means of interrupts, the system measures the water flow in liters per time unit from my meter.
- Every hour, the water consumption in liters is stored in a table.
- Every day at midnight, the value is stored to measure daily consumption (consumption at time t minus the consumption at previous midnight).
- Each year on December 31st at midnight, the consumption value is stored too. This allows for easy calculation of annual consumption (consumption at time t minus the consumption on January 1st).
- Measurements are stored in 24-hour slices. This allows for checking water consumption even in unusual time slices. For example, my water softener cleans its resin at 2:15 AM every 15 days.
- Every 15 minutes, the consumption over the last 24 hours is calculated. If it exceeds the alarm threshold (750 l / day in my case), an audible alarm is triggered, a virtual indicator on the Arduino Dashboard is switched on, and an email or a push notification is sent via the Arduino Cloud. All this to ensure that you are alerted in time!
- Similarly, the presence of two successive time slices without water consumption (often the case at night) is checked. If this condition is not met, a water leak may be suspected, and alerts are sent.
- The only maintenance operation required is to periodically enter the real value shown on the water meter. The difference between

the measurements made via the reed relay and the actual value shown by the meter is very slight. I readjust it approximately every two months.

Connect to Arduino Cloud

One can dismiss the classic Arduino IDE on your computer when working with Arduino Cloud [4]. The online compiler is sufficient for comfortably developing an application, and the successive and transparent updates significantly enhance the development environment. Since Ota (Over-the-Air) update is native in this environment, one can choose to either download the software via a USB port (mandatory for the first download) or remotely, which is very convenient.

There are many tutorials on the internet to learn how to master the cloud, but overall, it's not so complicated. Everything is free if you don't exceed five variables to refresh on the application. This is not a lot, but sufficient for the water leak detector if one is satisfied with limited display: daily consumption, two tables of time slices, annual consumption, and meter value.

For my part, I subscribed to a Maker subscription for €5.99 per month (but there are special offers) because I use this application for numerous projects. In the free version, one is limited to two applications. With my Maker subscription, my data is kept for three months, and I can download a CSV file containing the timestamped data.

If you want to have all the features I have developed (including statistics on operation and connection-with-reset commands), I recommend subscribing to at least one month of the Maker plan to try it out.

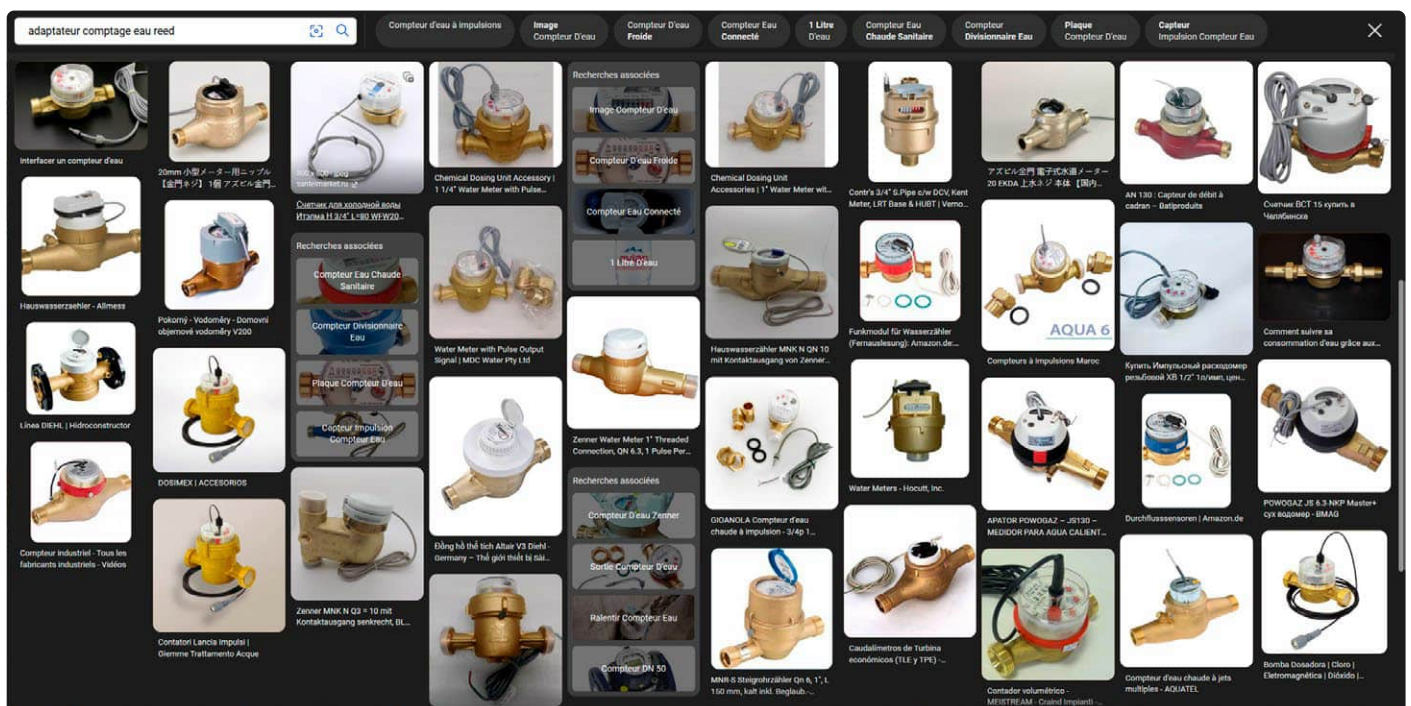


Figure 2: Reed sensors for all types of meters are readily available.

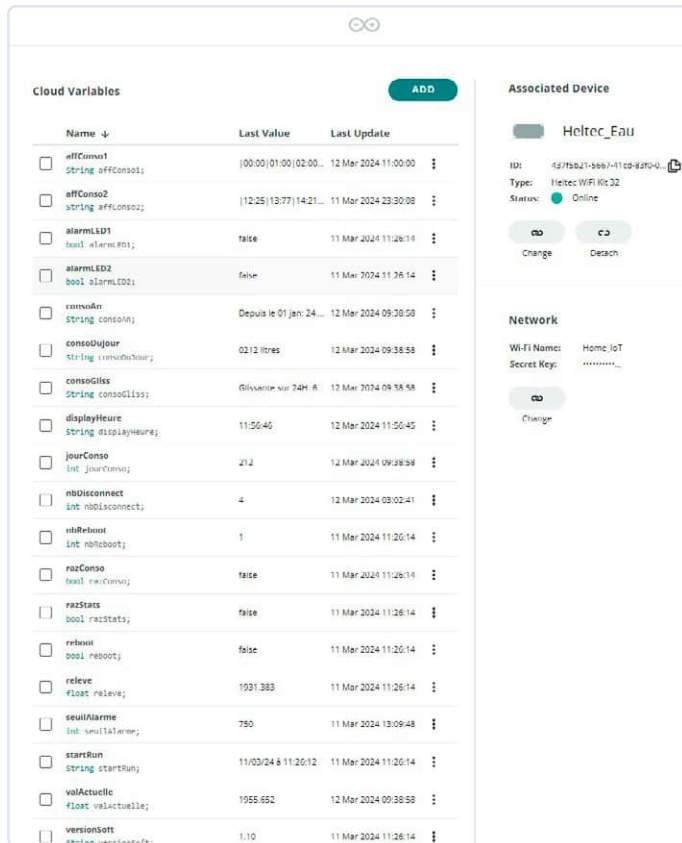


Figure 3: Step 2 — configure a “Thing.”



Figure 4: Here's what you should get after defining all the variables.

I optimized the connection reliability as follows:

- If the Wi-Fi connection fails, the system detects it and attempts to reconnect.
- If the connection to Arduino Cloud is lost (which happens occasionally), the program detects it and, after several reconnection attempts, the CPU is rebooted.

I also managed the local time as follows:

- When the system is connected to Arduino Cloud, it periodically retrieves the UTC time from the cloud.
- Since I need the local time including daylight saving time changes, the ESP32's local time is updated taking these parameters into account.

Step 1: Add a Device

This step prepares your system for the cloud environment by assigning it a Device ID and a Secret Key. Start by selecting *A third-party device* in the Setup Device menu, then *ESP32*, and then *Heltec WiFi Kit 32* from the drop-down list. Look carefully, because the list may not be sorted alphabetically, but it's there. Name it. The cloud then provides you with the two variables that must be saved properly using the downloadable PDF.

Step 2: Configure a “Thing”

Next, you need to associate the variables that you want to display or modify (be sure to respect the case!), see **Figure 3**.

You must define the types `bool`, `int`, `float`, `string` upon creation, as well as its *Read & Write* attribute. Read & Write is for a variable that can be modified in the mobile app, such as the alarm threshold. *Read Only* is for values that are only to be displayed, such as the alarm status. Leave *Variable update Policy - On change* for all variables.

You must also indicate the SSID and password from the *Network* menu, as well as the previously retrieved Secret Key. **Figure 4** shows the result of this step.

Step 3: The Sketch

Go to the *Sketch* tab and choose *Open full editor*. You are now in the development environment.

Load the libraries required for this project: *SSD1306.h*, *OLEDDisplayUi.h*, *TimeLib.h*, *math.h*, and *Preferences.h*. I placed them in the *Libraries* folder. Simply upload them using the arrow in the *Libraries* tab.

Download the source code from this project's page on Elektor Labs [5]. Open the file *Elektor_Thing_mar12a.ino* in the classic Arduino IDE or in a text editor to easily copy the program to the Arduino Cloud IDE. All you need to do is copy and paste the source code into the first tab.

Create the *images.h* tab using the + on the main bar, and copy and paste the contents of my file. You must also complete the *Secret* tab, as shown in **Figure 5**.

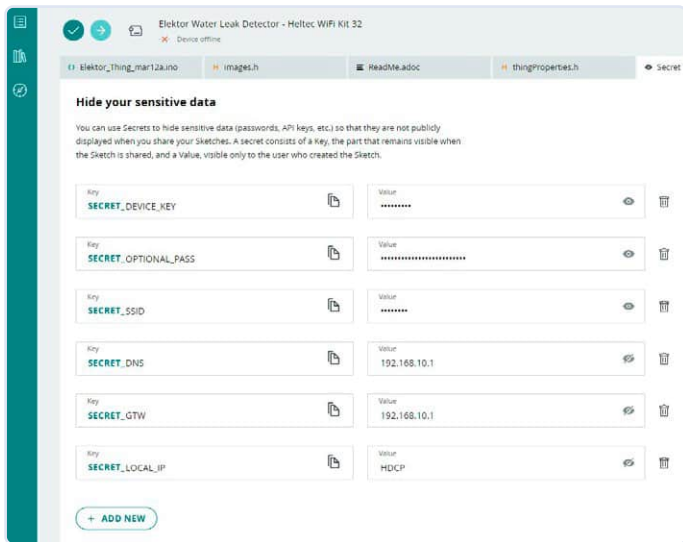


Figure 5: Complete the Secret Tab page like so.

Compile the program using the corresponding button. If everything is okay, you can download the program to your board (after installing Cloud Agent, as suggested by the IDE). Compiling is very fast — nothing like the classic IDE.

Note: I have grouped several parameters in the **Secret** tab. In addition to the default `SECRET_SSID`, `SECRET_PASSWORD`, and `SECRET_DEVICE_KEY`, you can define network parameters (IP, Gateway, DNS). If the IP address is *dhcp* or *DHCP*, then the Wi-Fi connection is established using DHCP.

The Dashboard

Creating a Dashboard like mine is simple (**Figures 6a and b**). Tap on + **Dashboard** in the main menu. You can rename it. Then press **Add**. Select the type of widget you want to display. Then all you must do

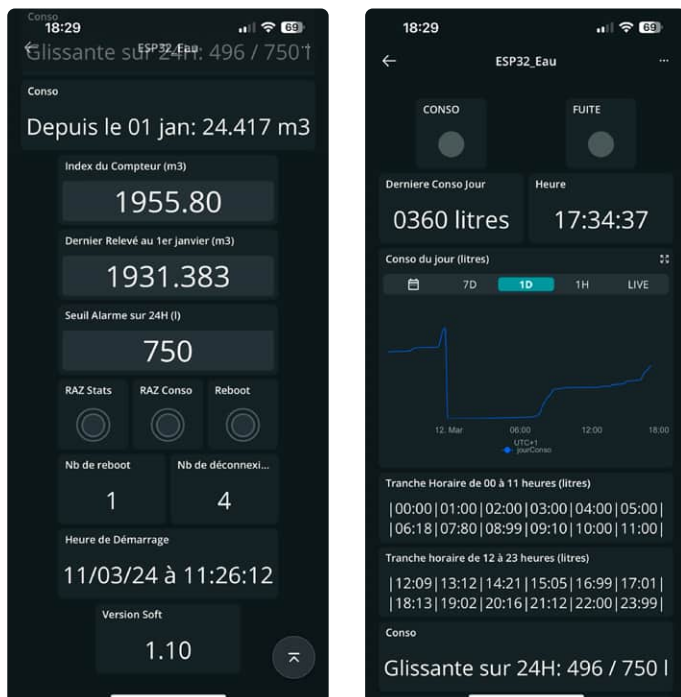


Figure 6a and b: This is what I see on my mobile!

A Bug?

Recently, I noticed an issue on my iPhone, but not on my tablet. In the Arduino dashboard, it is very complicated to enter a new value for the water meter count, the last reading on January 1st, or the alarm threshold. For some reason, the new value is immediately erased before you even have the chance to tap the **Done** button.

To work around this problem, which appeared after an update of Arduino Cloud, simply press the reboot button, enter the new value, and wait for the system to restart. The new value will then be taken into account.

This issue may have been solved by the time you read this article.

is **Link Variable** to assign a value to this widget. Validate with **Done** and repeat this procedure for all the variables you wish to view on the dashboard.

I assigned the variable `jourConso` also to a graphical widget to have a history of the water consumption.

Once all variables are assigned, select the edit function (two crossed arrows) to move and resize your widgets.

Click on the phone icon to have a different layout on your mobile. Creating a dashboard is fun and takes only a few minutes to get a pretty impressive result.

Email and Push Notifications

Again, nothing could be simpler. Just go to the **Triggers** page in the main menu. Click on + **Trigger** and then link the variable `alarmLED1` (for example), and choose to send an email or a push notification, or both.

The Circuit

I reused the same circuit (**Figure 7**) as used in some other projects, which you can find on my Elektor Labs page [6] (ESP32 Thermo-stat, VMC Regulator, Fuel Measurement, Linky Analyzer and Load Shedding, NTP Server).

I placed my device near my patch panel, which is backed up by a UPS, but you can also connect a rechargeable lithium battery (the Heltec module has a connector for this) which will be charged automatically and maintain power for about 20 minutes. Since the data is saved periodically, only the water consumption of the last 15 minutes is lost if you don't have electrical backup.

The water meter is almost 40 m away from my house. Therefore, to ensure the reliability of pulse counting, I preferred to use an optocoupler instead of an RC filter to block interference. Not only does it provide galvanic isolation, but it also filters out parasitic pulses that are too weak to light up the optocoupler's LED. About 10 mA at 3.3 V is required to light the LED properly. As a positive side effect, this much current also avoids oxidation of the reed relay's contacts. A 100 nF capacitor is sufficient to filter the connection to the reed sensor.

Two LEDs provide some status information. The blue LED1, connected to IO25 (SENSOR_LED), lights up for about 100 ms for each pulse received on IO23. When red LED2, connected to IO17 (CONNECT_LED), lights up, it means that the system is no longer connected to Arduino Cloud.

Software

The major difficulty of the project was not to miss pulses from the water meter. Therefore, great care was taken when writing the interrupt routine (**Listing 1**).

On each interrupt, the blue LED lights up. The global variable `Last_interrupt` records the time of the interrupt, allowing the main loop to turn off the LED about 100 ms later (flash).

The variable `indexCounter` is of type `volatile int`. It contains the number of pulses, i.e. the number of half-liters of water consumed. This variable can only be modified within the interrupt service routine using the commands `portENTER_CRITICAL_ISR(&mux)` and `portEXIT_CRITICAL_ISR(&mux)`.

My sensor sends two pulses per liter. This is configurable in the software as 1-2-4- pulses per liter. Depending on the characteristics of the sensor, the value is divided by 1, 2, 4, 8, etc. (using bit shifting) in the main loop:

```
Total_Counter = indexCounter >> pulsParLitre;  
// 2 pulses / l
```

The variable `indexJour`, also a `volatile int`, corresponds to the variable `jourConso` in Arduino Cloud. It is not updated if the value is modified manually in the Arduino dashboard. Like `indexCounter` it is divided by `pulsParLitre`.

Every 15 minutes

- The variables `indexCounter` and `indexJour` are saved in non-volatile memory (NVS).
- An array `conso[24]` contains the water consumption for each hour (for display reasons, only 99 l/h can be counted). This allows checking for a leak: if there aren't two consecutive hourly periods with a consumption of zero, it means water is continuously flowing. In this case, an alarm is triggered, and the variable `alarmLED2` is set to 1.
- We also check the total water quantity consumed since midnight. If it exceeds a threshold defined by the user, overconsumption is assumed. In this case, an alarm is triggered, and the variable `alarmLED1` is set to 1.
- In case of an alarm, a push notification and an email are sent via Arduino Cloud.

Every 60 minutes

The consumption for the next hour is reset to zero, and the screen is turned on for a period defined by the screen saver (5 min, see below). The small OLED screen allows displaying three pages that scroll every five seconds. This allows displaying network characteristics, time, different consumption counts, and, if applicable, the type

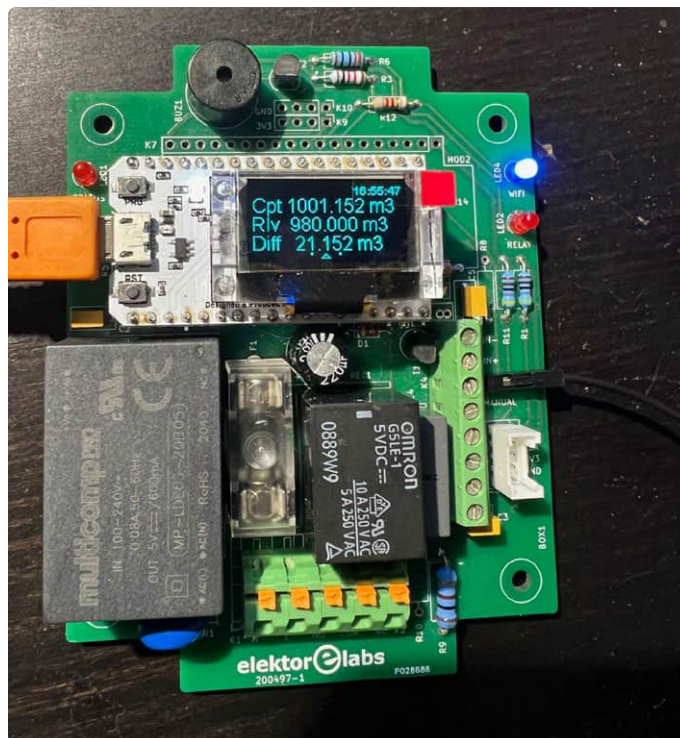


Figure 8: The electronic board assembly sporting the Heltec ESP32 module.



Figure 9: The board assembly in its enclosure.

of triggered alarm. To display the 24-hour consumption on two lines only, the `formatData()` routine is used.

Some additional functions allow:

- Resetting all hourly periods to 0 (normally only used during tests).
- Checking if Arduino Cloud has disconnected (which, of course, does not lead to counting errors). If after 600 s (10 minutes) the system still cannot connect to Arduino Cloud, a network problem



is assumed, and the system reboots. The number of reboots can be viewed, and at each reboot, the time and date of reconnection are indicated.

- On December 31st at midnight, the annual water consumption is reset to zero, and the new water consumption counter value (`Last_Counter`) is saved.

To check that everything was working as intended, I placed an impulse counter in parallel, and observed no discrepancy between what I measured and what the meter displayed.

Since I use an OLED screen that tends to age quickly when on continuously, I also integrated a screen saver routine to make the screen turn off after five minutes. It turns back on in case of an alarm or if a key connected to the system is pressed.

My Installation

For the hardware, I used the board that I had developed for an ESP32 Thermostat [7], see **Figures 8 and 9**. The advantage of this board is that the 230 VAC mains power supply is integrated onto the board, and all connections are made using 3.81 mm terminal blocks.

Successful Monitoring

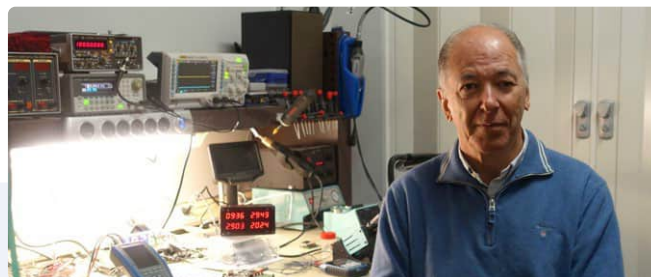
The water leak detector described in this article has been operational for more than one year now. Up to now, I have not noticed any bugs because I have been regularly updating the system to apply improvements and corrections.

A few months ago, a toilet situated outside the house started to leak very quietly. The system has detected a continuous leak of about twenty liters per hour from midnight until 7 AM. I noticed the alarm when I woke up. Without the detector, this event could easily have turned into an environmental and financial disaster.

The system also lets me see if my water softener has cleaned its resin at 2 AM and how much water is used for that. Similarly, it can show if it is malfunctioning.

The water leak detector has become a valuable asset of our home electric and electronics system. ◀

230720-01



About the Author

Passionate about electronics and technology since his earliest years, Yves Bourdon studied engineering at the INSA of Lyon. Even before completing his studies, he founded his first company, ERIM, in 1981. In 1991, Yves created the first French PC with an Intel 286 processor that had a footprint of just 10 cm × 10 cm. After selling his companies in 2009, Yves devoted much of his time to supporting innovative companies in their development. Among other roles, he served as a volunteer president of Cap'Tronic (a public structure) for over 10 years. Now 66 years old, Yves is retired and spends much of his time in his electronics laboratory exploring ESP32-based applications.

Questions or Comments?

Do you have technical questions or comments about this article? Email the author at yb.electronique@orange.fr or contact Elektor at editor@elektor.com.



Related Products

- **LILYGO T-Display ESP32 Development Board (16 MB)**
www.elektor.com/19774
- **D. Ibrahim, *The Complete ESP32 Projects Guide*, Elektor 2019**
www.elektor.com/18860

WEB LINKS

- [1] D. Lafourcade, "Waterflow Monitor with ESP32", Elektor 7-8/2019: <https://elektormagazine.com/180694-02>
- [2] S. Romero, "Connected Projects, Simplified", Elektor Guest Edition 12-2022/1-2023: <https://elektormagazine.com/magazine/elektor-268>
- [3] Heltec WiFi Kit 32: <https://heltec.org/project/wifi-kit32-v3/>
- [4] Arduino Cloud: <https://cloud.arduino.cc>
- [5] This project on Elektor Labs: <https://elektormagazine.com/labs/esp32-water-leak-detector-connected-to-iot-arduino>
- [6] The author's projects on Elektor Labs : <https://elektormagazine.com/labs/13670/ybourdon/projects>
- [7] Y. Bourdon, "ESP32-Connected Thermostat", Elektor 9-10/2021: <https://elektormagazine.com/200497-01>

Crystals

Peculiar Parts, the Series

By David Ashton (Australia)

From natural quartz to advanced synthetic forms, crystals play a pivotal role in electronics, providing precise frequency control for devices such as microcontrollers and radios. Let's take a look at their practical applications in electronics, from anything computerized to radio systems.

One of the strangest parts for an electronics beginner to get to grips with is the crystal. Basically, a bit of quartz glass with electrodes plated on both sides, the crystal is one of the most used and most recognizable electronic components. You will find at least one on pretty much any board with a microcontroller on it, and also in any radio system. The common requirement of both these devices is that a stable frequency reference is needed — for microcontrollers, the clock signal with which it operates and synchronizes its communications to the outside world, and, for radios, to set the frequency at which it operates.

The Crystal Evolution

Prior to about 1925, most oscillators used tuned circuits — a combination of (usually) a fixed inductor and a variable capacitor. Even with good design, they were liable to frequency drift.

In 1880, Jacques and Pierre Curie discovered the piezoelectric effect — a crystal substance with electrodes on both sides will deform if a voltage is put across the electrodes. If used to provide positive feedback in an amplifier circuit, an oscillator will result. In World War I, Paul Langevin used the effect to generate ultrasonic frequencies.



Figure 1: A small 1975 PCB-mount disc crystal of a few MHz with the cover removed, and a larger bar crystal — 100 kHz — from 1959.

In 1925, Westinghouse used a crystal oscillator in its radio station, KDKA, and crystal control was soon the norm.

Up to World War II, natural quartz crystals were used, most from Brazil. Bell Laboratories developed a process to grow synthetic quartz crystals, and, by 1970, most crystals were synthetic.

Understanding Crystal Properties

A crystal is equivalent to a precise, very high "Q" tuned circuit. Crystals have serial (low-impedance) and parallel (high-impedance) resonant frequencies. Below 30 MHz, a frequency between the two is usually used, while above that serial resonance is the norm. Crystals have different properties, depending on which "cut" of the crystal is used, i.e., where the crystal is cut relative to its axes. There are many other variables, and crystal physics is fiendishly complicated. Some crystals, particularly for high frequencies, may be made to operate at a harmonic or "overtone" of their natural frequency — 3rd, 5th, or even 7th.

Crystals do exhibit temperature dependence and techniques are used to overcome this — temperature compensation in the oscillator



Figure 2: Various ancient and modern crystals. Top: older types. Middle (from left to right): HC6/U, HC49/U (one with a red PTC device used as an "oven") and HC49S low-profile through-hole types. Bottom (from left to right): 32,768 Hz watch crystals, HC49SMD, and other SMD types.

circuit, or even an "oven" to keep the crystal at a set temperature above ambient. "Pulling" may be used to set the frequency precisely — usually an adjustable serial or parallel inductor or capacitor which slightly alters the natural resonant frequency.

Crystals are an essential, though sometimes misunderstood, component of most modern electronic equipment. **Figure 1** shows the internal construction of some crystals, and **Figure 2** shows both some ancient and modern crystal packages. [◀](#)

240214-01

Questions or Comments?

If you have technical questions or comments about this article, feel free to contact the Elektor editorial team by email at editor@elektor.com.



About the Author

David Ashton was born in London, grew up in Rhodesia (now Zimbabwe), lived and worked in Zimbabwe, and now lives in Australia. He has been interested in electronics since he was "knee-high to a grasshopper." Rhodesia was not the center of the electronics universe, so adapting, substituting, and scrounging components were skills he acquired early (and still prides himself on). He has run an electronics lab, but has worked mainly in telecommunications.



Universal Garden Logger

A Step Towards AI Gardening

By Gamal Labib (Egypt)

Smart irrigation, controlled by sensors, cares for water conservation and helps recover plants in stress due to inadequate scheduled watering. Moving to intelligent irrigation based on machine learning would be a step even beyond that. In this article, I present a data logger that marks the first step toward garden intelligence.

Source: Adobe Stock

Automating garden irrigation relieves owners from the burden of daily manual watering activities. Despite the automation, if the water distribution is inefficient, discolored spots can still appear in lawns, and plants can still show signs of stress.

Smart irrigation has been the focus of manufacturers and researchers for decades. Gathered data from sensors embedded in the garden presented the magic wand to activate or deactivate watering schemes as required to mitigate irregularities in plant conditions. The most popular sensor adopted for such purpose is the one that measures the soil moisture below the plant roots. Dry soil conditions would activate the irrigation process until the roots are well-hydrated [1].

Artificial intelligence (AI) takes the irrigation process into another dimension. Several approaches are possible. In this example [2], a simple color camera is used with machine learning techniques to identify if the plant is in a stress condition due to watering deficiency. Another example [3] adopts machine learning for analyzing the humidity, temperature, and pH [4] sensor readings in order to predict the soil's watering requirements.

In general, machine learning (ML) applications rely on the availability of a large quantity of data records relevant to the system, to be used to train the model. New data, that was never seen by the model, can then be fed to the model, which in return provides useful information about it and suitable actions can then be taken accordingly. The data used in ML applications can have different formatting, but one of the most popular formats is comma-separated values (CSV). When planning for AI gardening, we need to focus on data gathering. This is why I included a wide variety of sensors covering a wide range of gardening parameters.



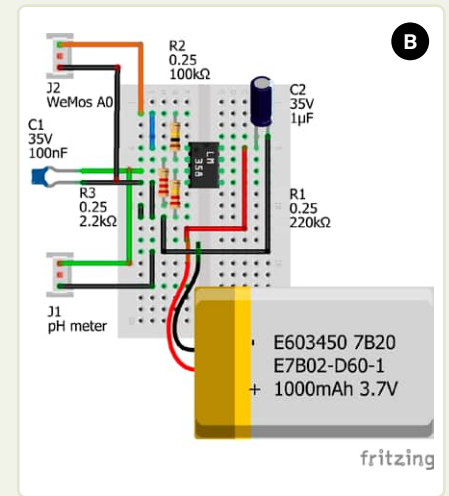
Interfacing a pH Meter

Soil pH fluctuates over long periods, requiring weeks and sometimes months to show measurable changes, especially when being re-adjusted by the farm owner. So, I considered the daily tracking of soil pH as a secondary target that may be attained manually or as an add-on to this project. When I looked at the available digital soil pH sensors in the market, I came to the conclusion that they are somehow expensive, especially those suitable for interfacing with Arduino projects.

The cheapest sensor I had available at the time of setting up this project was the manual one I already had for my garden, which amazingly required no batteries to operate (see **Figure A**). The sensor has two electrodes made from different metals, and the soil acts as the electrolyte between them. Such a setup forms an electrochemical cell, which is of course quite weak, but strong enough to drive the



small analog meter. Using an opamp and a small battery, I was able to amplify this signal and feed it to the analog input of the WeMos, alongside the other sensors (**Figure B**).



A “universal” data logger could produce a CSV file containing ambient readings of temperature and humidity, sunlight exposure, soil moisture, pH and NPK (nitrogen-phosphorus-potassium fertility and nutrient measurements), etc. It could also include the volume of water reaching the soil, which allows measurement of the efficiency of sprinklers used to distribute irrigation water. Unfortunately, the pH and NPK digital sensors are too expensive to experiment with. However, I managed to interface a cheap analog pH sensor to make it work with micro-controllers, as can be seen in the text frame, **Interfacing a pH Meter**.

Building the Project

The garden logger is built around the WeMos D1 Mini, a module based on an ESP8266 microcontroller. Having the capability of Wi-Fi connectivity, it supports the logger integration with IoT at home. The current implementation of the logger contains four analog sensors: a TEMENT6000 light sensor for measuring the plants’ sun exposure, a rain sensor that detects rainfall (which doesn’t happen often in my region) and which I am also planning to use to adjust the sprinklers’ coverage areas, a water level sensor for measuring collected rainwater or water from sprinklers hitting the data logging location, and a soil moisture sensor for root-dehydration detection. The setup includes a single digital sensor — the DHT11 temperature and humidity sensor — to measure ambient conditions around the logger. It’s worth noting that some of the analog sensors have an additional digital output that’s set to ON or OFF, depending on the threshold preset by the onboard potentiometer. The logger is shown in **Figure 1**.

Since the ESP8266 has only one analog input interface (A0), and I wanted to connect many analog sensors to it, I used an analog multiplexer chip, the 4052, to enable the connection of up to 8 analog sensors to the logger. The analog multiplexer required two GPIO interfaces, pins 16 (D0) and 14 (D5) of the D1 Mini, to select one of the four analog sensors and connect it to the microcontroller’s analog input.

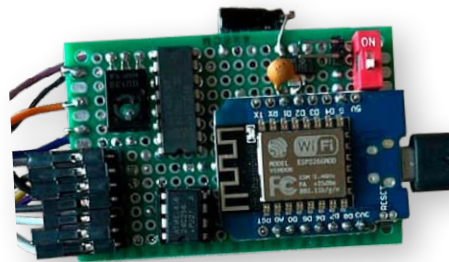
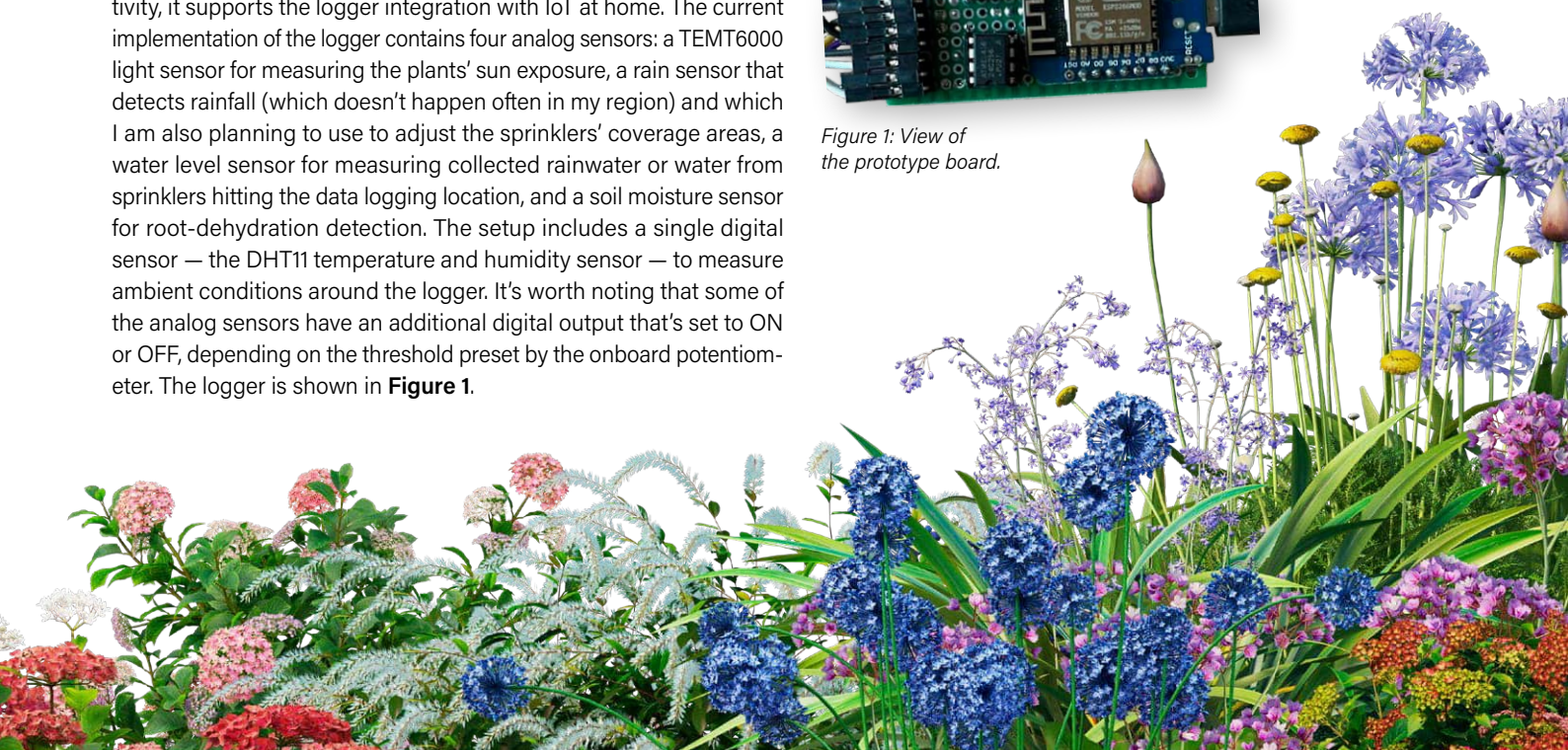


Figure 1: View of the prototype board.



Since the logger is powered by a battery and is intended to run for extended periods, energy conservation becomes paramount. During the intervals in which logging is disabled, I managed to switch off the sensors by cutting off their ground connection using a BD139 NPN transistor capable of sinking 1.5 A, whose base is driven directly from the GPIO15/D8 pin of the D1 Mini and can tolerate a voltage up to 5 V. Setting that pin to High will put the transistor into saturation and enable the sensors. Otherwise, it is set to Low to save energy.

Since the logger is meant to be universal, I chose to report the plain analog readings of the sensors, which gives the user the freedom of calibrating the sensors and interpreting the values with the most flexibility. The rain sensor was an exception where I relied on both its analog and digital outputs as described later on. I used GPIO13/D7 pin to monitor its digital output.

Figure 2 illustrates the wiring of the project components. The rain and soil sensors are fully passive, and thus require driver modules to produce their measurable analog output, while the rest of the sensors are self-sufficient. The complete bill of materials is included at the end of the article. **Figure 3** shows the physical wiring of project components prior to being fitted in the logger enclosure.

When fitted, the ambient sensor is accessible from the bottom side of the casing to be as close as possible to the soil environment but still protected from irrigation or rainwater. On the top side of the logger, the light and rain sensors are exposed to the environment for best monitoring of sunlight and falling water drops. A glass hatch would be required to isolate the light sensors from the elements. As for the water level and the soil moisture sensors, they are wired to the logger using 20 cm-long cables to facilitate positioning.

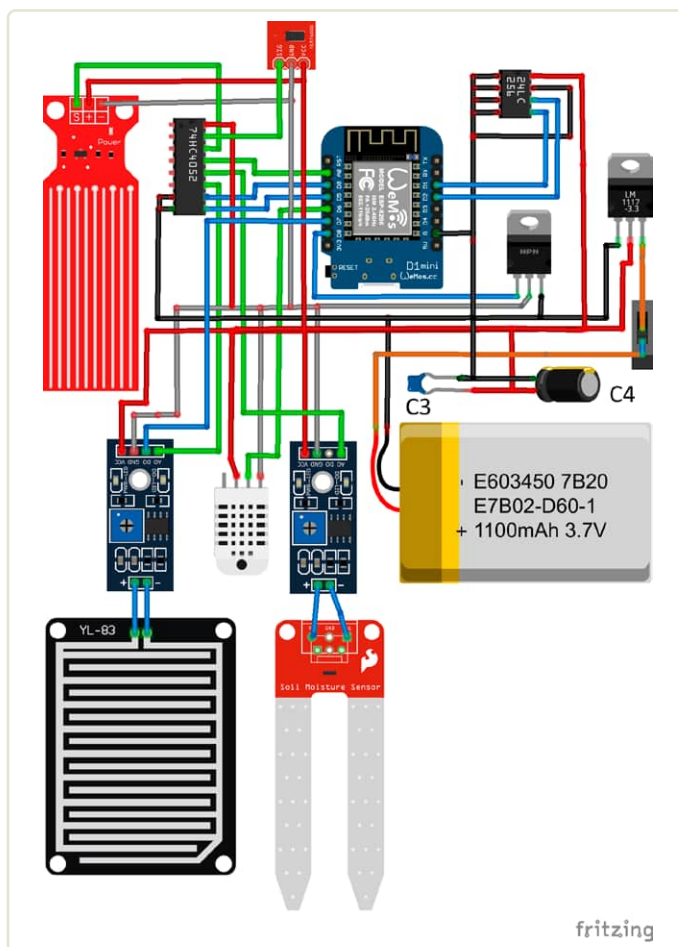


Figure 2: Garden logger circuit wiring.

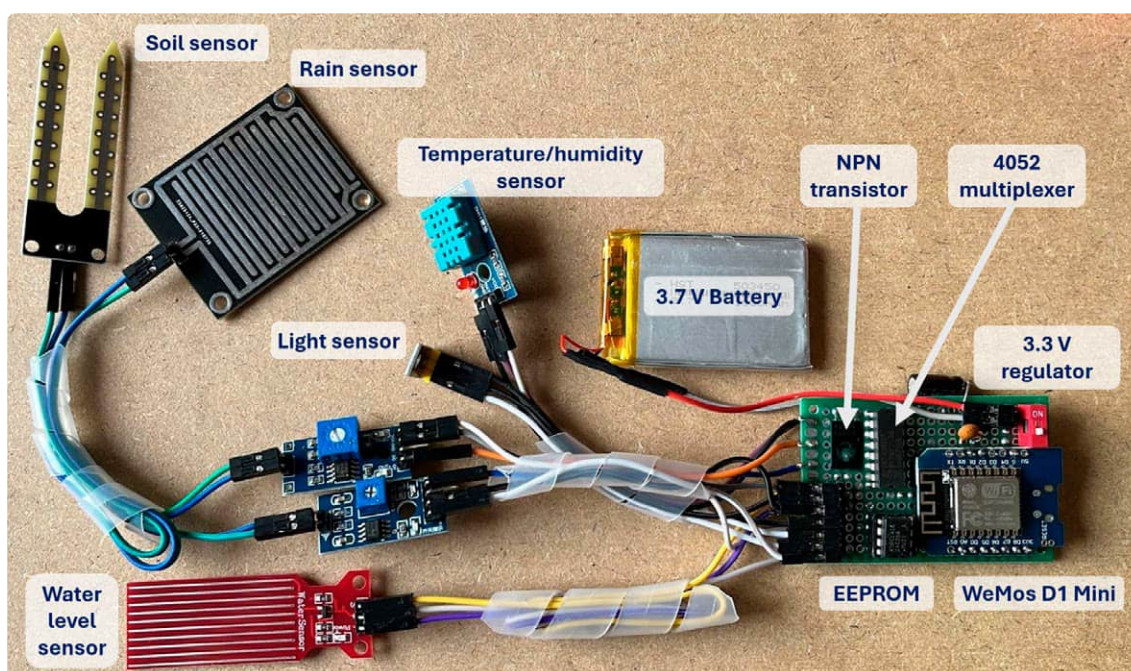


Figure 3: All the parts for the project, connected outside the enclosure.

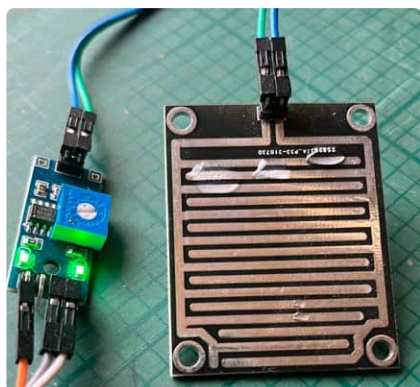
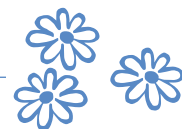


Figure 4: The rain sensor.

13:55:15.169	-> Analog output: 538	→ New reading
13:55:16.187	-> Analog output: 538	} Readings induced by water viscosity
13:55:17.186	-> Analog output: 540	
13:55:18.190	-> Analog output: 538	
13:55:19.169	-> Analog output: 538	
13:55:20.165	-> Analog output: 538	
13:55:21.159	-> Analog output: 538	
13:55:22.187	-> Analog output: 537	
13:55:23.189	-> Analog output: 537	
13:55:24.182	-> Analog output: 537	

Figure 5: Interpreting measurements from the rain sensor.

The logger uses a 3.7 V lithium battery as a power source. This battery has a moderate 1,100 mAh capacity and its dimensions snugly fit the project enclosure. Since the WeMos D1 operates on 3.3 V, I had to use a low-dropout (LDO) regulator that steps down the battery voltage by a mere 0.4 V to match the WeMos specification. I chose a regulator rated at an output current of 800 mA to be able to drive the logger components. The user needs to choose a battery with a capacity that complies with the intended working duration of the logger.

Estimating Water Amount

Measuring the amount of water received at the point of interest in the garden is a critical metric when it comes to resolving issues of plant stress. I used the water level sensor to be able to take such measurements. This is a traditional method of collecting the falling water, either from rain or from sprinklers, into a funnel discharging into a graded container. The theory behind this simple arrangement is that the height of water level L in a regular container, say a cylinder with known cross-section area A , corresponds to volume V using the formula $V = L \times A$. So, when writing the Arduino sketch, I consider area A as a defined constant, and I record the progress of collected water volume against time using the water level sensor readings for L .

The rain sensor (**Figure 4**), on the other hand, takes a different approach for water calculations. The sensor works using the same principle as the water level sensor, that is, the more water falling onto the sensor, the less electrical resistance it produces. However, it has different shaped metal traces that enable it to detect the drops of water falling at some instance in time. The problem is that, due to the viscosity of water, drops tend to accumulate between the metal traces.

To avoid this problem, I installed the logger in the garden with its casing tilted such that the metal traces are at an angle and directed toward ground to help the water drops slide off, and get rid of drops that have already been accounted for. This also required momentary detection of water drops when they hit the sensor and discarding the overwhelming readings that follow as water slips off the sensor and makes further contact. Another problem is that water drops get blocked at the bottom longitudinal horizontal metal trace and stay there indefinitely.

Here, the digital output of the sensor driver circuit comes to the rescue. Tuning the potentiometer shown in the previous figure helps neutralize the effect of that situation, and the digital output will then indicate no rain (logic High). Remaining now is the effect of dynamic movement of water drops as they slide down the metal traces. Experimenting with the rain sensor shows slight changes to the analog output of the sensor. I accounted for this phenomenon by discarding the trailing readings close to the one that produced a new value. This action is illustrated in **Figure 5**, where the analog output of a single drop of water shows 538 while the trailing readings are almost the same as the drop slides down by gravity.

When the rain sensor is used to adjust fixed or rotating sprinklers, the system must take into account the two main categories of lawn sprinklers: fixed ones, which irrigate a fixed radius around them, and rotating ones, which use a portion of the water's flow energy to activate the rotating mechanism. To account for both types of sprinklers, the microcontroller's firmware must be able to distinguish between a continuous uniform flow of water and intermittent water hits.

Handling Logged Data

The garden logger is equipped with a 32 KB external EEPROM chip to hold the sensed data throughout the logging duration. I mounted the chip on a DIP socket rather than soldering it to the prototype board to enable its replacement with higher storage capacity chips, if necessary. There's also the ESP8266's internal EEPROM, which can be used for the same purpose but with a limited storage size of 4 KB. On the other hand, extensive writing to the internal EEPROM has the adverse effect of shortening the lifetime of the ESP device, so I don't recommend using it for data logging.

Each record of logged data is formatted as comma-separated values. I assumed that all measurements would be in integer format, since this application did not require the precision of fractional values. So, the code casts all `Float` measurements into the `Integer` data type, then uses the `String()` function to convert them to strings (`strVal`). Cascading a `strcat(strVal, ",")` function for sensed data embeds the comma between data items in the record, and the emerging string is pre-headered with the record's timestamp.

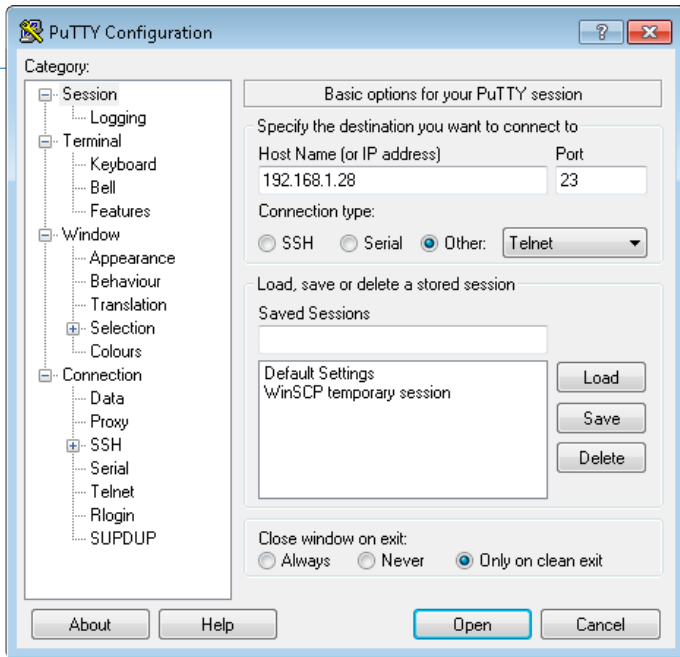


Figure 6: PuTTY configuration (1/2).

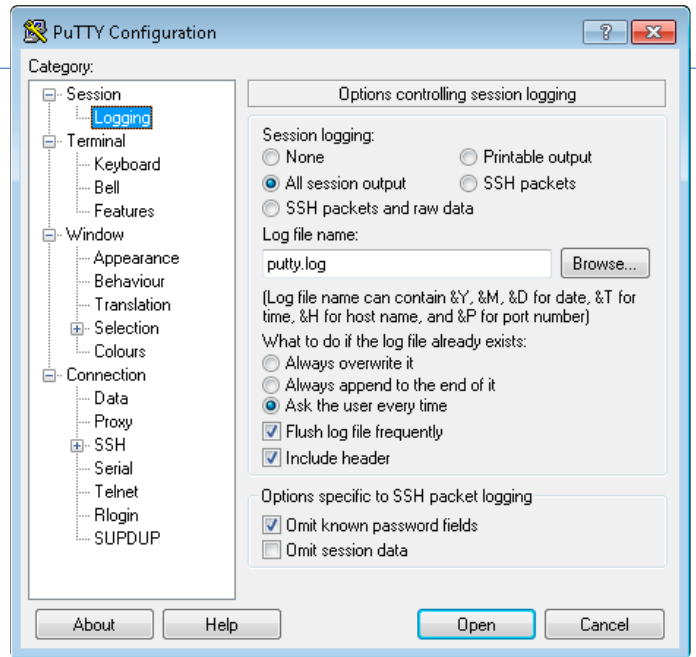


Figure 7: PuTTY configuration (2/2).

The project does not include an RTC module to keep precise timestamps. Instead, it relies on contacting an internet NTP server when it reboots at midnight every day to get the time details, then it uses the `millis()` function to calculate the current time of measurement and generates a timestamp from that. The structure of a sensed data record is: *timestamp, temperature, humidity, light, collected volume, dropped volume, soil moisture*.

There are several possible approaches regarding transmission of the recorded values to a computer. One way could be to have a webserver running on the datalogger itself. Alternatively, it is possible to program the microcontroller so that, once the records are written to the external EEPROM, they are sent to a PC in batch mode; a Telnet client such as PuTTY can be used to save the records to a CSV file. A third way is to send the records to the PC as soon as one line of CSV data is constructed. I implemented the later mode of operation in the current version of this project. **Figure 6** illustrates PuTTY's configuration window on the PC.

The procedure to follow is rather simple: Make sure that the garden logger and the PC are connected to the same wireless network, enter the logger's IP address in the *Host Name* box, set the port to 23, and select the Telnet protocol from the drop box. Prior to clicking **Open** to start the telnet shell, we need to set up the logging of the data communicated from the logger. This step is illustrated in **Figure 7**. You need to select *All session output* and *Ask the user every time* to make sure not to overwrite an existing log file from previous sessions. Then, the location of the destination CSV file must be chosen, using the **Browse** button.

Now, you can click on **Open** to start the telnet shell. Once started, the shell starts collecting logged records coming from the logger, as depicted in **Figure 8**. It's possible to terminate the logging session by typing "C"; the logger will respond with the message "bye bye" generated from the `telnetAction()` function. Sensor values are also available on the serial port as they are assembled in the log records.

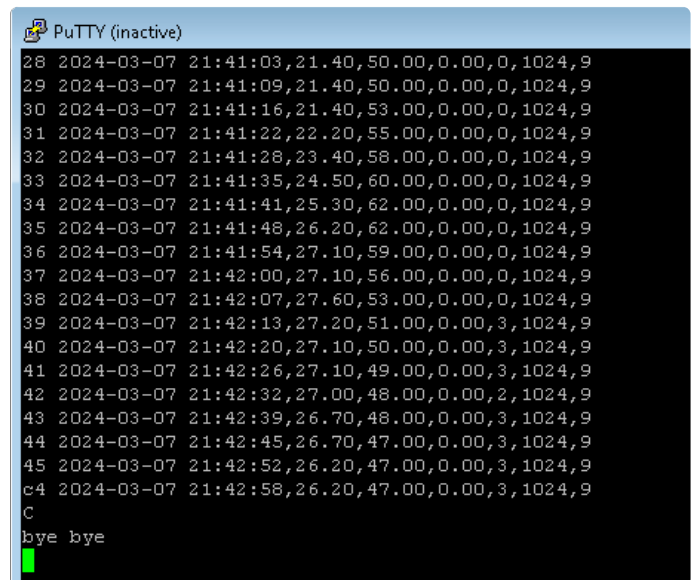
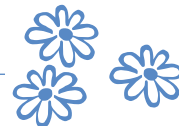


Figure 8: Logging data to the PC in the telnet terminal window.

They can be read from the serial port, as shown in Serial Monitor in the Arduino IDE in **Figure 9**.

The Code

The project is programmed using the Arduino IDE. The program is composed of a main .ino file, *Garden.ino*, alongside a set of five auxiliary .ino files. Each one of the additional files consolidates the functions driving a common peripheral. *Timing.ino* contains the functions related to the NTP server, *telnet-streaming.ino* deals with the formatting of the records and sending them to the PC, *analog-sensors.ino* contains four functions dedicated to the light, rain, water volume, and soil moisture analog sensors. Finally, *digital-sensors.ino* contains a single function that reads the ambient humidity and temperature, whereas *external-eeprom.ino* contains the functions used to read and write the EEPROM chip.



The screenshot shows the Serial Monitor window with the following output:

```
rain detected!
Water Level (cm): 657 Water Volume (cm3): 6
Soil Moisture (unscaled): 1024 Soil Moisture (scaled): 9
Humidity: 47.00% Temperature: 26.20°C 79.16°F Heat index: 26.10°C 78.99°F
Raw ADC data: 22.00 Volts: 0.07 Lux: 46.79
Analog Input: 380 Digital Input: 0
-62
rain detected!
Water Level (cm): 667 Water Volume (cm3): 6
Soil Moisture (unscaled): 1024 Soil Moisture (scaled): 9
Humidity: 47.00% Temperature: 26.20°C 79.16°F Heat index: 26.10°C 78.99°F
Raw ADC data: 27.00 Volts: 0.09 Lux: 57.43
Analog Input: 385 Digital Input: 0
-62
rain detected!
Water Level (cm): 649 Water Volume (cm3): 6
Soil Moisture (unscaled): 1024 Soil Moisture (scaled): 9
Humidity: 47.00% Temperature: 26.20°C 79.16°F Heat index: 26.10°C 78.99°F
Raw ADC data: 27.00 Volts: 0.09 Lux: 57.43
Analog Input: 387 Digital Input: 0
-62
rain detected!
Water Level (cm): 639 Water Volume (cm3): 6
Soil Moisture (unscaled): 1024 Soil Moisture (scaled): 9
Humidity: 47.00% Temperature: 25.80°C 78.44°F Heat index: 25.66°C 78.19°F
Raw ADC data: 28.00 Volts: 0.09 Lux: 59.55
Analog Input: 399 Digital Input: 0
-61
rain detected!
Water Level (cm): 629 Water Volume (cm3): 6
```

Figure 9: Output data of each sensor in the Arduino IDE's Serial Monitor.

This structured style of coding makes it easy to follow up and debug each functional part of the project and facilitates reusability. **Listing 1** shows the `loop()` section of the sketch encompassing only ten statements, six of which are function calls discussed earlier. The listing also depicts the part of the `setup()` section by which I control the multiplexing of the four analog outputs from the sensors to the single analog input pin of the D1 Mini. Also in that section, I dedicate a GPIO to control the transistor that's used to turn the sensor on and off, in order to save power. See the complete code at [5].

It's worth noting that the ESP8266 has an analog-to-digital converter (ADC) with a 10-bit resolution, making it capable of transforming analog input into a value in the range 0 to 1,023. Many existing projects convert this value to an 8-bit representation using mapping functions, such as `map(analogVal, 0, 1023, 0, 255)`, thus shrinking the range of values to 0 to 255. I refrained from doing such mapping in order to deliver sensor readings as accurately as possible.

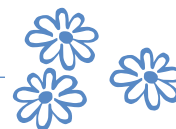


Listing 1. Main loop.

```
void loop() {
    delay(2000);
    digitalWrite(SENSORS, HIGH); // enable sensors power
    delay(1000);
    senseAmbient(); // read ambient humidity & temperature
    senseLight(); // read ambient light intensity
    senseRain(); // read rain status
    senseWater(); // read irrigation water level
    senseSoil(); // read soil moisture level
    telnetAction(); // assemble logged data and send to telnet app
    digitalWrite(SENSORS, LOW); // disable sensors power
}

// this is part of setup()
pinMode(MUXA, OUTPUT); // part of 2-bit analog sensor selection address
pinMode(MUXB, OUTPUT); // part of 2-bit analog sensor selection address
pinMode(SENSORS, OUTPUT); // sensors GND control pin
pinMode(DHTPIN, INPUT_PULLUP); // ambient DHT sensor feed
pinMode(RAIN, INPUT); // rain status sensor digital feed
digitalWrite(SENSORS, LOW); // disable sensors power

// see [5] for the complete code
```



Component List

Microcontroller module

WeMos D1 Mini (ESP8266), <https://amazon.com/dp/B07W8ZQY62>

Sensors

Water level sensor, <https://amazon.com/dp/B09J2NK21Y>

Rain sensor, <https://amazon.com/dp/B01DK29K28>

Soil moisture sensor, <https://amazon.com/dp/B083Q8DFHX>

Ambient Sensor DHT11, <https://amazon.com/dp/B0CCF2C2CF>

Light sensor TGMT6000, <https://amazon.com/dp/B00L8DW8L2>

Miscellaneous

C3 = 100 nF, ceramic

C4 = 470 μ F, 10 V

24LC256 EEPROM (Mouser)

LM1117MP-3.3-NOPB 3.3V regulator (Mouser)

BD139 transistor (Mouser)

CD4052BE analog multiplexer (Mouser)

Lithium battery 1,100 mAh 3.7V,

<https://amazon.com/dp/B06WRRQGR6>

Project box 85x50x21 mm, <https://amazon.com/dp/B0B4VGGN6J>

Double-sided FR-4 PCB 40x60 mm,

<https://amazon.com/dp/B0968F3748>

Final Remarks

Intelligent irrigation based on machine learning requires a large amount of sensor data in order to produce an accurate irrigation model predicting the best irrigation setup under different ambient conditions. This project facilitates the logging of environmental conditions that a garden will be subjected to. An upgrade to the project may consider the recording of the status of plants of interest, using a camera or some other type of sensor, and add this information to the log records of this project. This would enable us to build an efficient irrigation model using supervised machine learning techniques. ◀

230629-01



About the Author

Gamal Labib has been an enthusiast of embedded systems for two decades and is currently a mentor (at codementor.io). He holds an MEng and a PhD in IT. Besides writing for technical magazines, he is a visiting associate professor at Egyptian universities and a certified IT consultant.

Questions or Comments?

Do you have questions or comments about this article? Email the author at drgamallabib@yahoo.co.uk, or contact Elektor at editor@elektor.com.



Related Products

- > **WeMos D1 mini Pro**
www.elektor.com/19185
- > **ESP8266 ESP-01 Wi-Fi Module**
www.elektor.com/17326
- > **SparkFun Sensor Kit**
www.elektor.com/19620



WEB LINKS

- [1] Irrigation project example: <https://circuitdigest.com/microcontroller-projects/automatic-irrigation-system-using-arduino-uno>
- [2] AI plant monitoring: <https://smellslike.ml/posts/tf-microcontroller-challenge-droopthereitis/>
- [3] Arduino-based smart irrigation project: <https://tinyurl.com/2kkbc2zx>
- [4] Information about soil pH: <https://esf.edu/eis/eis-soil-ph.php>
- [5] Downloads: <https://elektormagazine.com/230629-01>

Analog 1 kHz Generator

Sine Waves with Low Distortion

By Alfred Rosenkränzer (Germany)

A high-quality sine wave signal at 1 kHz is essential for measurements on audio electronics. Digital equipment still does not quite come close to analog sine wave generators. As always in the high-end sector, ready-made devices are not exactly inexpensive. With the circuit proposed here, building such a generator yourself offers significant cost advantages without having to compromise too much on signal quality.

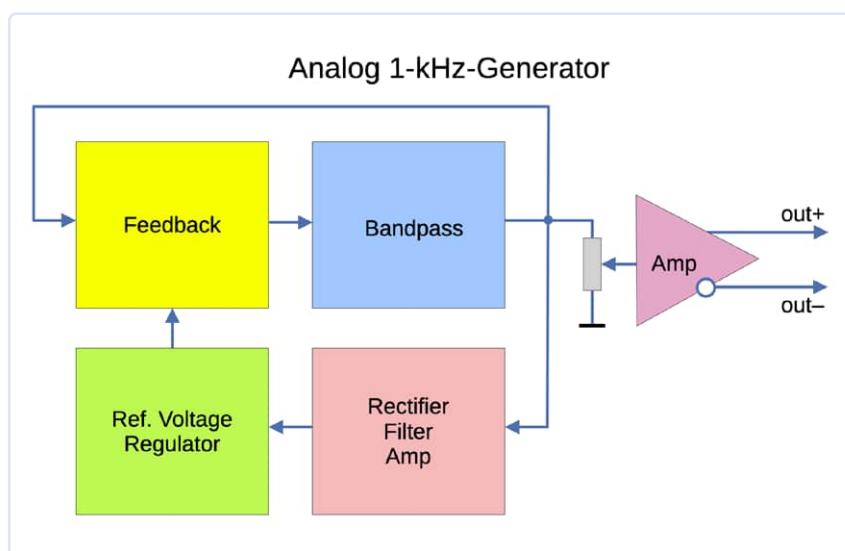


Figure 2: The block diagram of the 1 kHz generator.

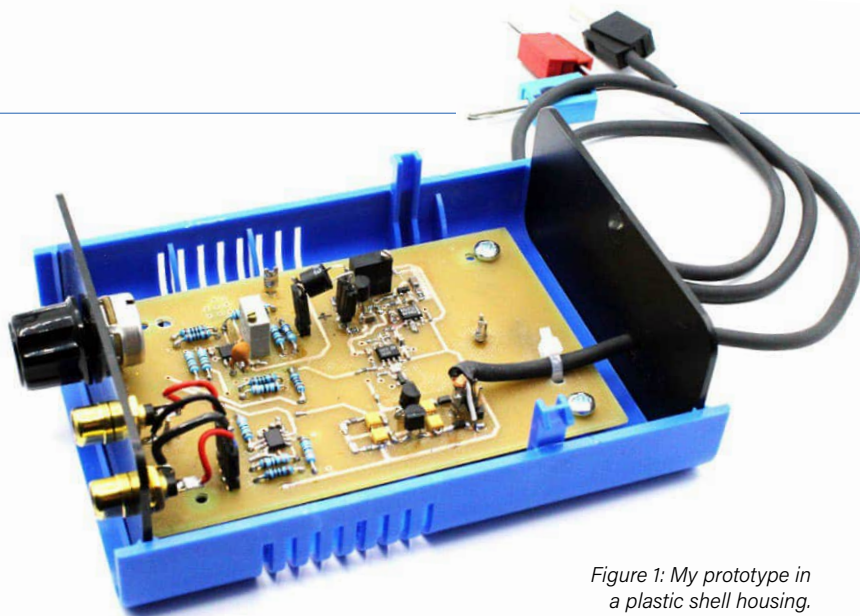


Figure 1: My prototype in a plastic shell housing.

My review of the QA403 from QuantAsylum [1] shows that digital sine wave generators are quite usable. Unfortunately, this company has retired its analog model QA480 — a supplement to the QA403 (see blog at [2]). This device combined an analog 1 kHz generator with a suitable notch filter and a high-quality 12 dB amplifier. Amplitude and notch filter could be controlled via USB from a PC. But that is history. According to the blog, the discontinuation is partly due to the excessive effort required for adjustment.

The circuit was based on the design of another company [3] and was fortunately published in this blog. As there is currently no equivalent device available to buy, I have slightly revised the circuit diagram and developed a suitable circuit board. The prototype can be seen in **Figure 1**. Power is supplied via an external, balanced ± 15 V power supply unit. For the sake of simplicity, I have implemented the amplitude adjustment with a potentiometer. So there is neither USB nor software and therefore no interference signals from digital electronics.

The Circuit

The block diagram in **Figure 2** shows five functional units. The actual oscillator consists of a band-pass filter, which is made to oscillate by positive feedback. The amplitude is stabilized by a control circuit, whose actual value is obtained from the output signal by rectification and filtering. A reference voltage is required for a stable signal. The output amplifier provides a differential signal of adjustable amplitude.

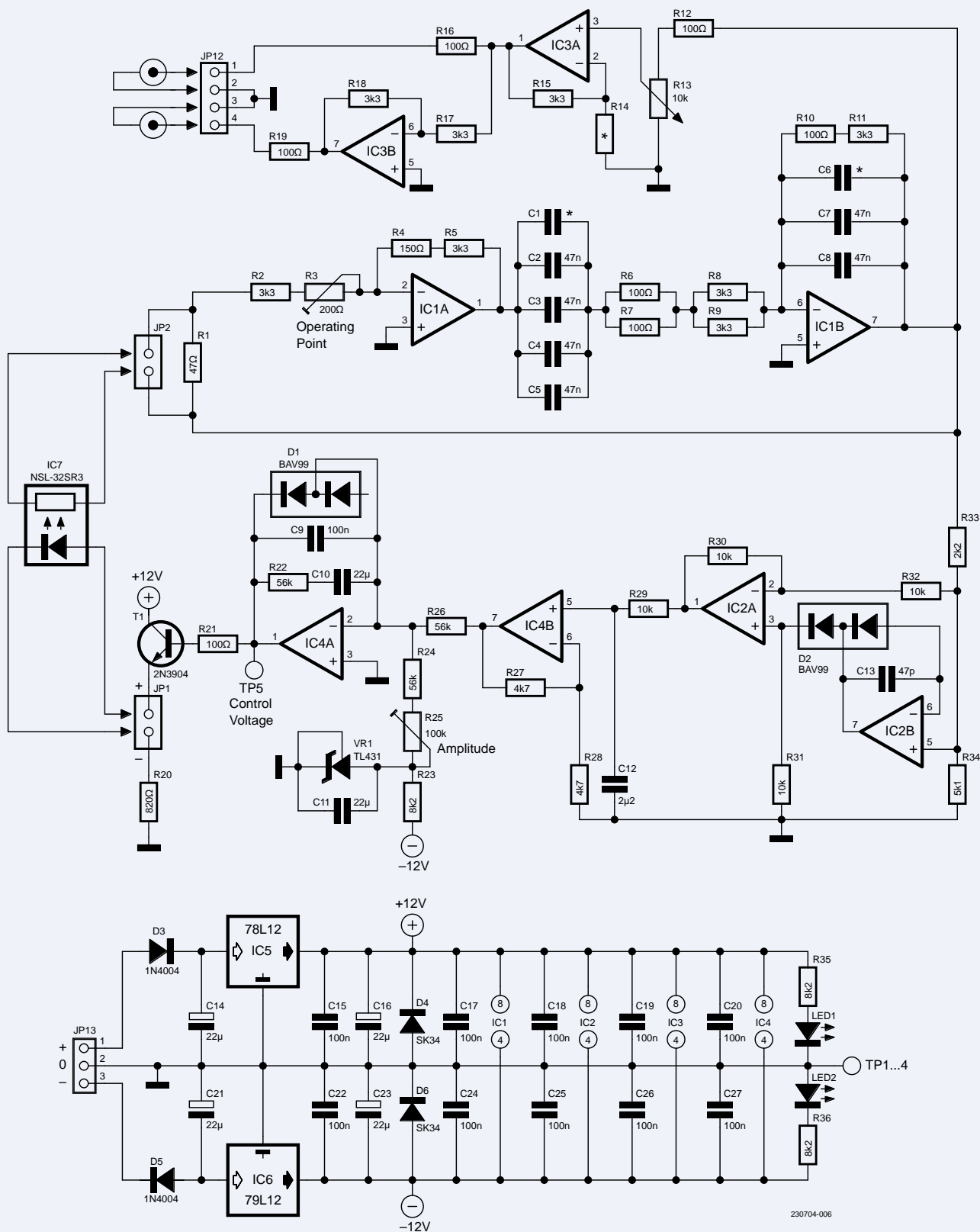


Figure 3: The circuit only requires four dual op-amps, two voltage regulators, and one optocoupler.

The circuit in **Figure 3** shows how the functional blocks are implemented in actual electronics.

Band-Pass

The band-pass is realized by connect-

ing IC1B with R6 ... R11 and C1 ... C8. Its center frequency determines the oscillation frequency. The quality of the components used here influences the distortion of the generated sine wave signal. Either ceramic COG capacitors in SMD housings or wired

film capacitors can be used. The circuit therefore shows twice the number of capacitors, of which only half, i.e. SMDs or leaded film capacitors, are to be fitted. On the circuit board, you can clearly see that only two capacitors are actually connected in parallel



Component List

Resistors

Unless otherwise specified:

MELF 0204 or thin film SMD 1206, 1%
 R1 = 47 Ω
 R2, R5, R8, R9, R11, R15, R17, R18 = 3k Ω
 R3 = 200 Ω , multi-turn trim pot, vertical,
 RM 1/10"
 R4 = 150 Ω
 R6, R7, R10, R12, R16, R19 = 100 Ω^*
 R12, R16, R19 = 100 Ω
 R13 = 10 k, potentiometer, linear
 R14 = optional*
 R20 = 820 Ω , SMD 0603
 R21 = 100 Ω , SMD 0603
 R22, R24, R26 = 56 k Ω , SMD 0603
 R23, R35, R36 = 8k Ω , SMD 0603
 R25 = 100 k Ω , multi-turn trim pot, vertical,
 RM 1/10"
 R27, R28 = 4k Ω , SMD 0603
 R29, R30, R31, R32 = 10 k Ω , SMD 0603
 R33 = 2k Ω , SMD 0603
 R34 = 5k Ω , SMD 0603

Capacitors

C1, C6 = optional*, COG, SMD 0603
 C2, C3, C8 = 47 nF, film, RM 2/10"
 (alternative to C4, C5, C7)*
 C4, C5, C7 = 47 n, COG, SMD0805
 (alternative to C2, C3, C8)*
 C9, C15, C17...C20, C22, C24...C27 = 100 nF,
 X7R, SMD 0603
 C10, C11 = 22 μ F / 16 V, SMD 1206
 C12 = 2 μ 2 / 16 V, SMD 1206
 C13 = 47 pF, COG, SMD 0603
 C14, C16, C21, C23 = 22 μ F / 25 V, Tantal
 Elko, SMC-B

Semiconductors

LED1, LED2 = LED, SMD 0805
 D1, D2 = BAV199, SOT23
 D3, D5 = 1N4004
 D4, D6 = SK34, DO214AC
 T1 = 2N3904, SOT23
 IC1, IC3 = OPA2211*, SOIC8
 IC2, IC4 = TL072, SOIC8
 IC5 = 78L12, TO92
 IC6 = 79L12, TO92
 IC7 = NSL-32SR3*
 VR1 = TL431, TO92

Miscellaneous

JP1, JP2 = 2-pin header, LP 1/10"
 JP12 = 4-pin header, LP 1/10"
 JP13 = 3-pin header, LP 1/10"
 Circuit board 230704-01
 Knob for pot R13
 Audio sockets*

* See text

and fitted at the input of IC1B (plus C1 if you want to adjust the frequency). In the feedback path, there is even only a single capacitor (plus possibly C6 for frequency adjustment). The pads of the resistors have the SMD1206 design, which is easy to solder by hand. MELF204 resistors, which are similar in quality to wired metal film versions, also fit on these pads. Thin-film versions can also be used instead of MELF versions.

The capacitance in the input branch is twice as large as in the feedback path. On the other hand, the resistors at the input are only half as large as in the feedback path. This measure should lead to lower noise, at least mathematically. It is advisable to measure the capacitors precisely and connect identical specimens in parallel if possible. This makes it easier to fine-tune the frequency later using C1 and C6 or R6/R7 and R10. The band-pass has a gain of about -1 at the center frequency.

Positive Feedback

For stable oscillations, the (controllable) positive feedback of the circuit around IC1A (R1 ... R5) must also have a gain of approximately -1 . If the gain is too low, the circuit will not oscillate, and if it is too high, the signals will be so large that they will be limited by the possible voltage swing of the op-amps. A constant amplitude with an unlimited, sinusoidal signal shape can only be achieved by controlling the positive feedback.

The readily available optocoupler NSL-32SR3 from Advanced Photonix [4] serves as the control element. The current through the LED on the input side influences the resistance of the integrated LDR on the output side. This optocoupler provides the usual electrical isolation between input and output.

Amplitude Measurement

Approximately 70% of the output signal from IC1B reaches the input of IC2B via the voltage divider R33/R34. This attenuation prevents strange transient behavior after switching on when the oscillator is briefly driven to the amplitude limit.

The circuit of the precision rectifier built with IC2 is described very well in a reference design paper from TI [5]. The rectified

signal is integrated or "averaged" by the low-pass filter consisting of R29 and C12 and then amplified twice by IC4B. If you want to calculate the corresponding values, a suitable online calculator [6] is helpful. IC4A works as a regulator thanks to its circuitry.

Amplitude Stabilization

The commonly available IC TL431 acts as the reference voltage source VR1. A stabilized voltage of -2.5 V drops across it to ground. C11 reduces the noise. The current from IC4B and R26 is compensated by R24 and R25 and the amplitude is adjusted with R25. D1 prevents the control voltage at the output of IC4A from becoming negative. T1 is controlled via R21. Its emitter drives the LED of the optocoupler via R20, whose output resistance changes the positive feedback. This closes the control loop. R26, R22, C9, and C10 determine its time constant.

Output Stage

Via R12, the sine signal reaches the potentiometer R13, which is used to adjust the output amplitude between 0 and the maximum value defined by R25. IC3A serves as an output amplifier. R14 can be used to set the amplification slightly higher if required. IC3B amplifies (inverting) with a factor of 1 and thus forms the negative output signal. The signal sockets can be RCA, BNC, TS audio jack, or XLR types as required.

Power Supply

The circuit is supplied with ± 15 V. Two voltage regulators generate stabilized ± 12 V from this. D3/D5 prevent damage caused by reverse polarity, and D4/D6 protect IC5 and IC6 from a latch-up when switching on. The two LEDs light up when the power supply is active.

Selecting the Components

The quality of the op-amps of IC1 and IC3 has a major influence on the signal quality. For this reason, I have used the slightly more expensive OPA2211 types here. Of course, the circuit also works with other, pin-compatible dual op-amps. You can use almost any suitable dual op-amp for IC2 and IC4. My prototype uses TL072.

The capacitors C1 to C8 also influence the distortion. As already mentioned, COG types in SMD housings or wired film capacitors

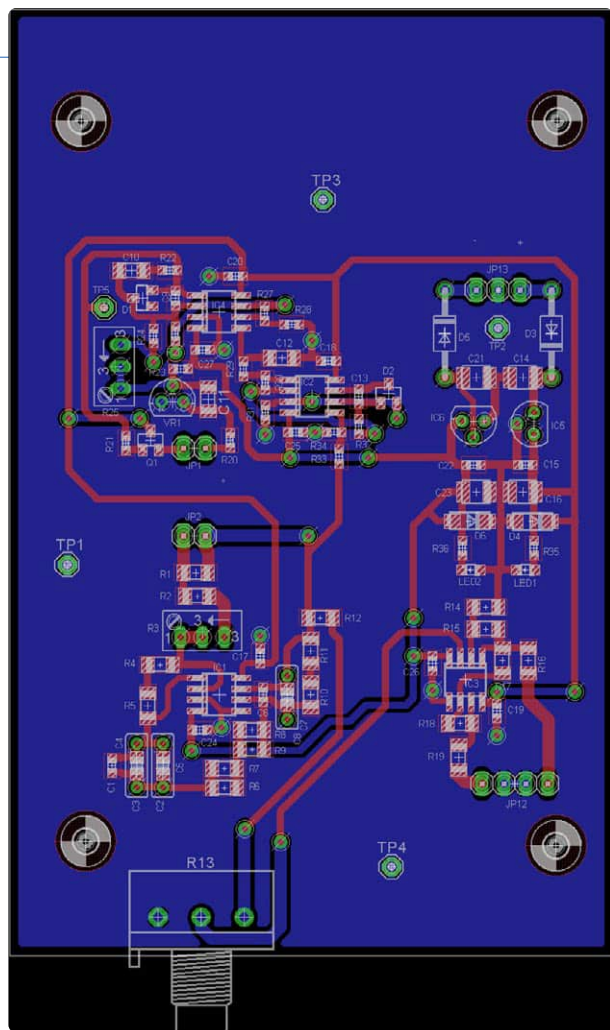


Figure 4: Layout of the board. The corresponding files can be downloaded in Eagle format at [7].

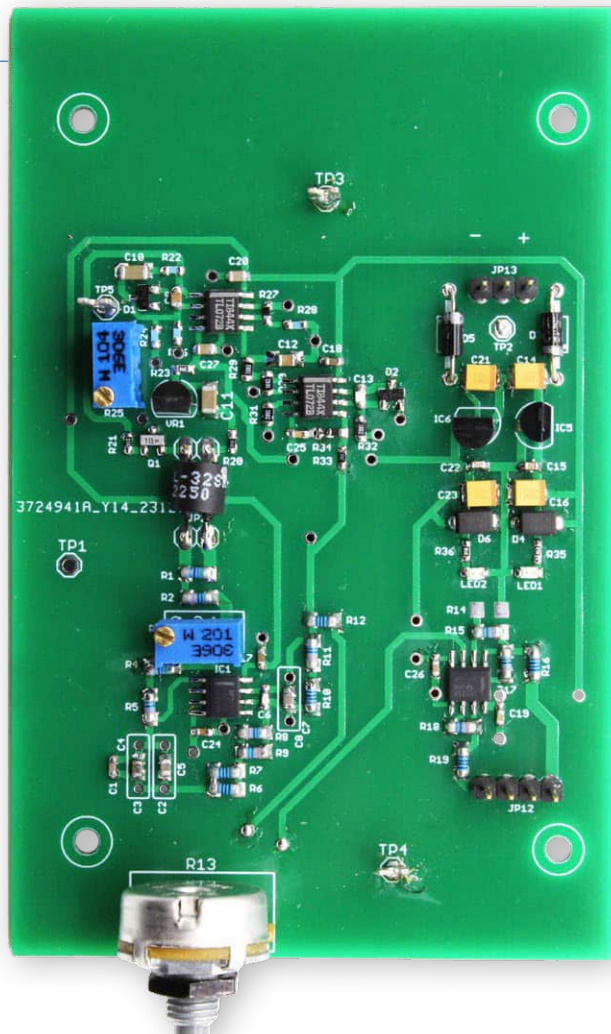


Figure 5: The fully assembled circuit board of my prototype.

should be used here. Note: A single “bad” capacitor according to audiophile criteria with XR7 dielectric or similar will ruin everything! The resistors in the signal path should be thin-film resistors of type 1206 or MiniMELF204.

Assembly and Commissioning

The optocoupler IC7 should not be fitted initially. Figure 1 shows that I have made it pluggable using two 2-pin pin headers/sockets. This makes it easier to test the circuit: If you short-circuit the pins for the integrated LDR and turn potentiometer R3, the oscillator should oscillate at a certain setting and generate a signal limited to 20 V_{PP}.

This allows you to check the function of the output stage, rectifier, and filter, including the amplifier. The frequency should be close to 1 kHz. The fine adjustment is carried out later at a controlled, stable amplitude.

The frequency changes slightly when the op-amps reach saturation. When the signal is limited, the output of IC4A should deliver 0 V. If the oscillator is stopped, it slowly increases to approximately +10 V.

The transfer characteristics of the optocoupler can vary considerably. I bought four of them and each one behaved differently. The current through the LED can be adjusted with a laboratory power supply and a series resistor, and then the resistance can be measured with a multimeter. An LED current of 3 ... 5 mA has proven to be a good operating point.

For further commissioning, you should connect a resistor with the measured value in parallel to R1 and then set R3 so that the oscillator just begins to oscillate. If necessary, you can adjust R4. If you remove the resistor and insert the optocoupler, the circuit should provide a sinusoidal signal with a constant and stable amplitude. R25 can be used to

adjust the desired amplitude to values of 1 ... 3 V_{PP}. Larger amplitudes will result in higher distortion.

The control time constant was deliberately chosen to be quite large. This means that the settling time is long, but the level is very constant afterwards. The operating point of the controller is adjusted with R3.

Frequency and More

The frequency can now be fine-tuned. You can change the capacitance by fitting C1 and C6 (C1 = 2 × C6) or adjust the values of R6/R7 and R10. The frequency is calculated using the usual formula $f = 1/(2 \times \pi \times R \times C)$. After frequency adjustment, it may be necessary to correct the operating point again. If you have a notch filter, the frequencies of the generator and the notch should be the same.

I have developed a suitable PCB for the circuit. **Figure 4** shows its layout and

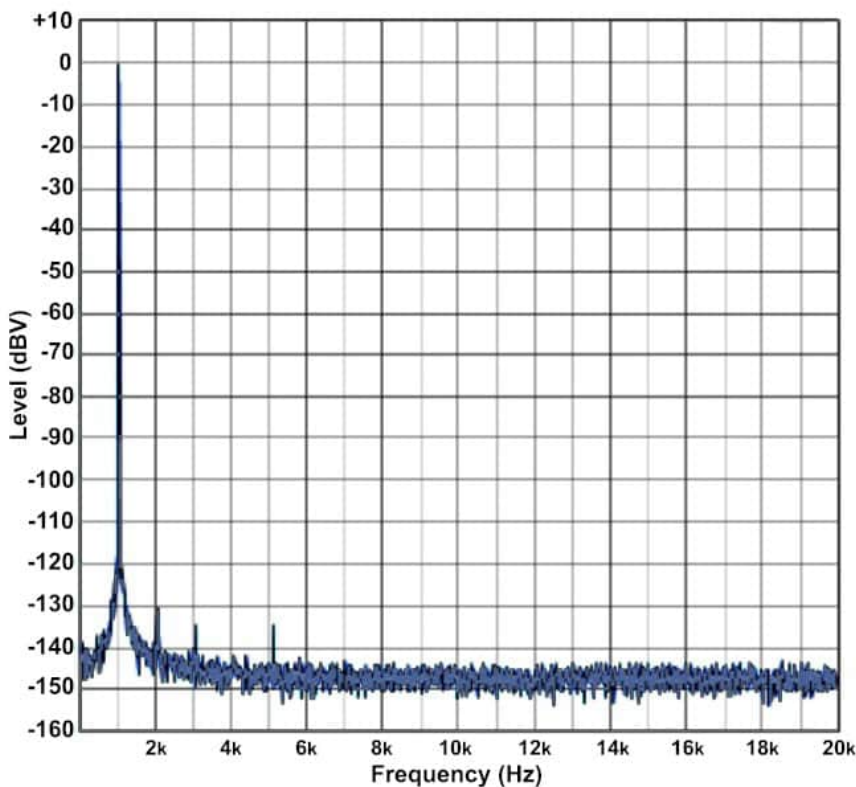



Figure 6: The spectrum of the output signal at a level of 0 dBV.

Figure 5 the assembled board. **Figure 6** shows the spectrum at the nominal level of 0 dBV. All harmonics have a level of less than -130 dBV: The output signal therefore has extremely low distortion. This spectrum was recorded using the APx555 audio analyzer from Audio Precision. The quality of the DIY signal generator is therefore quite respectable!

The component values described in the circuit work quite well with my prototype, but there is still room for optimization. I also have a few empty boards left. If you are interested, please contact me directly. 

Translated by Jörg Starkmuth — 230704-01



About the Author

Alfred Rosenkränzer worked for many years as a development engineer, initially in the field of professional television technology. Since the late 1990s, he has been developing digital high-speed and analog circuits for IC testers. Audio is his private hobbyhorse.

Questions or Comments?

Do you have questions or comments about this article? Email the author at alfred_rosenkraenzer@gmx.de or contact Elektor at editor@elektor.com.



Related Products

- > **QuantAsylum QA403 24-Bit Audio Analyzer**
www.elektor.com/20530
- > **FNIRSI 2C23T (3-in-1) 2-ch Oscilloscope (10 MHz) + Multimeter + Signal Generator**
www.elektor.com/20717



WEB LINKS

- [1] A. Rosenkränzer, "Comparing the QuantAsylum QA403 to the Gold Standard," 2023: <https://tinyurl.com/yudmy5sk>
- [2] Blog QA480 (retired): <https://tinyurl.com/yckdwe8e>
- [3] JanasCard website: <http://www.janascard.cz/aHome.html>
- [4] Optocoupler NSL-32SR3 (Mouser): <https://tinyurl.com/4c5c8nss>
- [5] Precision rectifier (TI): <http://www.ti.com/lit/ug/tidu030/tidu030.pdf>
- [6] Calculate AC voltage parameters: <https://tinyurl.com/vsrbpzbf>
- [7] Article web page: <https://www.elektormagazine.com/230704-01>

Miletus:

Using Web Apps Offline

System and Device Access Included!

By Dr. Veikko Krypczyk (Germany)

Web applications have become a standard in many areas today. They are executed in a browser and can therefore be run on almost all devices, including a Raspberry Pi. With the Miletus framework, these web apps can be packaged as native applications so that they can be executed offline. It also gives you access to the local system interfaces, for example to read and output signals via GPIO pins.

Applications for controlling hardware components that are connected to a PC or Raspberry Pi must be adapted to the respective operating system and require access to the system's interfaces. These are known as "native applications" and are created specifically for the target system and executed on it. On the other hand, there are web applications. A web application can be started on almost all systems because it runs in a browser. However, it also has some limitations compared to a native app. Offline use and access to the system inter-

faces are not readily possible. **Table 1** shows a comparison of the main characteristics of native vs. web applications.

For applications that are of particular interest to electronics engineers, arguments such as offline use, access to system interfaces, and the integration of individual drivers are likely to speak most in favor of a native desktop application. If you want to run this application on different systems, for example in Windows, macOS, and on a Raspberry Pi,

Table 1: Important characteristics of native vs. web applications.

Characteristic	Native application	Web application
User Interface	native	based on HTML, CSS, and JavaScript
Installation	necessary — executable files must be copied to the system	not necessary — the files are loaded by the browser via the internet
Updates	must be installed locally on each device	managed centrally on the server
Execution	directly on the system	in the browser
Cross-platform	no — a separate app must be created for each system	yes
Offline operation	yes	no
Access to device and system functions	yes	no
Access to locally installed databases, e.g. SQLite	yes	no
Control of specific hardware via drivers	yes	no
Access to file system	yes	limited

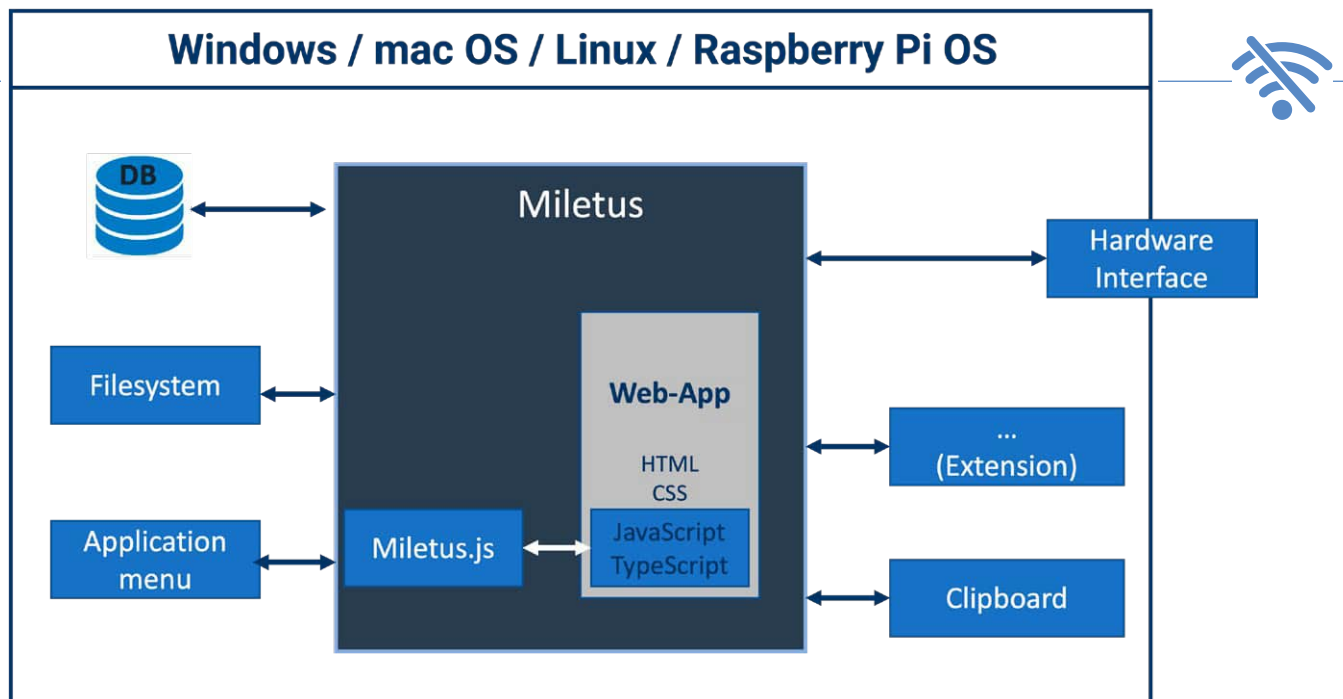


Figure 1: Architecture of a Miletus application.

then things quickly become very complicated. Each system has its own rules when it comes to creating the user interface, for example. In other words: three target systems, three applications, and therefore three times the development effort. In addition to the effort involved, the required know-how is usually also a problem, as programming a native Windows app is completely different from programming an application for the Raspberry Pi or for macOS. A web app, on the other hand, is created using HTML, CSS, and JavaScript and runs in a browser; in addition, powerful libraries such as Bootstrap simplify the design of the interface.

In this article, we present an interesting framework called *Miletus* [1]. With Miletus, you can package a new or existing web app into an application package for different target systems (including the Raspberry Pi) and run it like a native application. The application can then be used offline and has access to the system functions.

The Miletus Web Framework

Before we get to the practical use, we will introduce the most important features of Miletus. The applications created will work on Windows, macOS, and Linux operating systems, including Raspberry Pi OS. The Miletus API provides access to the system and device functions, the file system and, via drivers, to external hardware connected to the device.

From an electronics engineer's point of view, a particularly interesting feature is the fact that you can access the hardware interfaces of the Raspberry Pi, i.e. the GPIO, I²C, SPI, and UART ports and the memory buffer. In this way, almost all limitations of the web application are removed, and you still benefit from its cross-platform capability. For a better understanding, **Figure 1** outlines the architecture of an application based on the Miletus framework.

Embedding the web application in a native application package for the respective target system opens up new fields of application. These include, for example, applications for machine control or applications

from the Internet of Things (IoT). In both cases, signals must be sent and received via the hardware interfaces of the Raspberry Pi. Another special feature of the framework is the extensibility of the API. This is done via "shared libraries" — in Windows, for example, in the form of a DLL (dynamic linked library) file. If such a system library exists for the target system, such as a driver for specific hardware, then this library can be integrated into Miletus and its functions can then be accessed from the web app. Miletus uses the standard browser engine provided by the target system, i.e. in:

- > **Windows:** *WebView2*
- > **Linux/Raspberry Pi OS:** *WebKitGTK*
- > **macOS:** *WebKit (Safari)*

It can generally be assumed that the browser engines mentioned are already installed on the target systems, so that no additional provision by the application is necessary. This means that the application package to be distributed is not very large, i.e. the distribution of the app is accelerated, and it uses system resources sparingly. This is particularly important for execution on the Raspberry Pi. Because the system's existing browser is used, the applications also automatically benefit from functional and security updates to the browser.

However, some system requirements must still be checked on the target system and additional system libraries may have to be installed. The following libraries must be available on the target systems:

- > **Windows 32/64-bit:** The latest version of the Edge Chromium browser must be installed. In addition, the files *WebView2Loader_x86.dll* (32-bit) or *WebView2Loader_x64.dll* (64-bit) must be copied to the *System32* or *SysWow64* folder, respectively. These files are provided with the Miletus installation.
- > **Linux/Raspberry Pi OS:** The *GTK3* graphics interface is used, which can be installed via the command line using the command `sudo apt install libwebkit2gtk-4.0-dev`.
- > **macOS:** No further prerequisites are required.

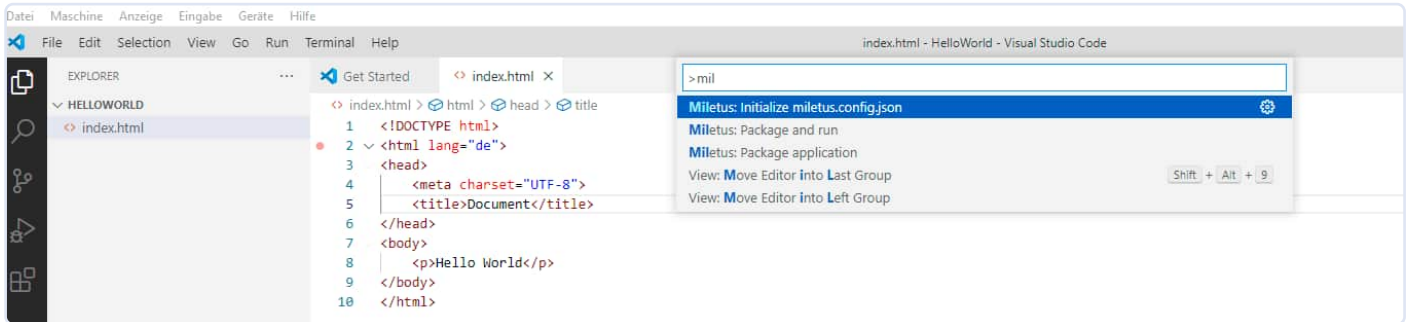


Figure 2: Commands for the Miletus Packager in Visual Studio Code.

Alternative: Electron

The Electron framework [2] is also used to package a web app into a desktop application. For example, the Visual Studio Code editor used here was created with it (i.e. this is actually a web app). In contrast to Miletus, Electron delivers its own browser engine, which means that the application packages are significantly larger. Also, Electron apps cannot be run on the Raspberry Pi, and the API cannot be extended.

Installation and First Experiments

What we need:

- **Web application:** For initial experiments, a single HTML file (*index.html*) that outputs only "Hello, world" text is sufficient. We also place a button to demonstrate how the Miletus API works (see below).
- **Visual Studio Code:** This editor can be downloaded from [3] for your own operating system.
- **Extension for Visual Studio Code:** This can also be downloaded from the Miletus page [1] and installed manually in Visual Studio Code (*vsix* file).
- **Packager (optional):** This can be downloaded for Windows, macOS, or Linux from [4] and installed. It allows the packager to be used via the command line.

A note for experienced web developers: If the npm package manager is installed on your system, you can also install Miletus via `npm install miletus`. The API is then installed on the development system and can be integrated into your own web projects.

Here is the *index.html* of our "Hello, world" example. The source code for the examples presented here can be found on the author's website at [5].

```
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script src="miletus.js"></script>
  <script>
    function save() {
```

```
      miletus.dialogs.showSaveDialog();
    }
  }
</script>
</head>

<body>
  <p>Hello World</p>
  <button onclick="save()">Save to File</button>
</body>
</html>
```

Now we have everything in place. If you have installed the extension for Visual Studio Code, the packager is set up automatically. The following additional commands are then available in Visual Studio Code (**Figure 2**):

- **Miletus — Package application:** Creates the application packages for the Windows, macOS, and Linux / Raspberry Pi OS target systems.
- **Miletus — Initialize config:** Creates a template for the configuration file *miletus.config.json*.
- **Miletus — Package and run:** Creates the packages for the selected target systems and starts the application on the development system. A Raspberry Pi cannot be used as the development environment; the application can only be executed on it after packaging.

Open the folder with the *index.html* file. You can open this HTML file in your browser for testing (**Figure 3**).

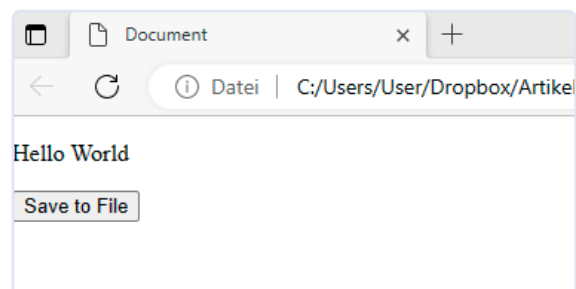


Figure 3: A web app with a button.



Now we want to package this “web app” into a desktop application. The Packager is controlled via the *miletus.config.json* configuration file. The name and version of the application, the target systems, the entry point for the application, and a few other things are specified here. The exact description of the configuration variables can be found in the documentation [6]. You can create the file manually or generate it automatically with Visual Studio Code. The latter is done using the command button (*[Ctrl + Shift + P]*) and the command, *Miletus: Initialize config*. Complete the queries, i.e., assign a name for the application, assign the start file (*index.html*), and select the desired target systems. The generated *miletus.config.json* file looks like this:

```
{
  "name": "Dialogs",
  "output": "output",
  "debug": true,
  "main": {
    "html": "index.html"
  },
  "target": [
    "win_ia32"
  ],
  "include": [
    "index.html",
    "miletus.js"
  ]
}
```

The application packages for the target systems are created from the information in the configuration file, for example, *win_ia32* for Windows 32-bit. This is done using the command *Miletus: Package application*. For Windows, this is an *exe* file, for macOS an *.app* folder and the associated entitlements file for the authorizations and for Linux an executable file and a *.desktop* file (Figure 4).

Now let's start the web app as a desktop application on the target system for the first time. To do this, the executable files must be copied to the target system, i.e. specifically:

- > **Windows:** Run the *.exe* file. If you are working with Visual Studio Code in Windows, you can run the application directly using the *Miletus: Package and run* command.
- > **Linux:** Copy the *Output* folder to the Linux system. Assign the rights to the files using the command: `sudo chmod -R 755 / [Path of the Application]`. Now start the application.
- > **macOS:** You must copy the application files from the *Output* folder, assign the rights, and then the application can be started. Please note: On an ARM-based macOS system, code signing of the app is necessary.

You can see the running “Hello, world” application in Figure 5.

Note: If you receive an error message or see an empty window, check the above requirements on your system.

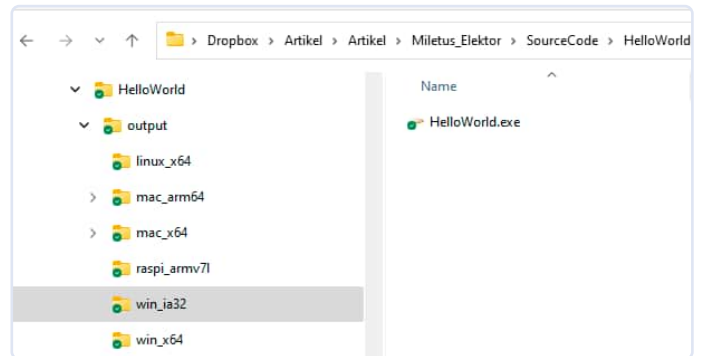


Figure 4: Application packages for the various target systems.

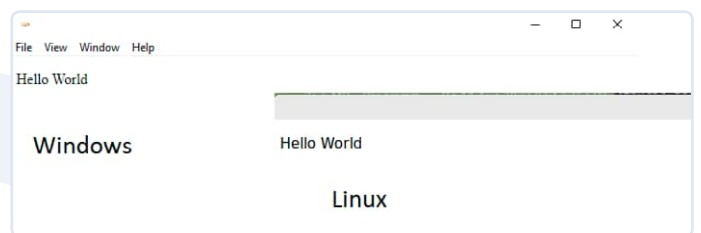


Figure 5: “Hello, world” app on Windows and Linux.

Using the Miletus API

A “Hello World” application does not yet show the use of the framework. We will demonstrate the framework with a simple example. The Miletus API offers the following functions, among others:

- > **Dialogs:** The Miletus API provides functions for displaying file dialogs (*showOpenDialog(...)*, *showSaveDialog(...)*), an error dialog (*showErrorDialog(...)*) and a message box (*showMessageBox(...)*). The functions are called in the JavaScript or alternatively in the TypeScript source code. First you have to include the corresponding library *miletus.js*. Then, for example, the *Save as...* dialog box can be called via:

```
miletus.dialogs.showSaveDialog()
```

For this purpose, this source code line is integrated into a function with the name *save()*, which in turn is called via a button's *onclick()* event. Figure 6 shows the call of the file dialog.

- > **File access:** Miletus provides methods for loading and writing text and binary files via the *loadTextFile(...)*, *saveTextFile(...)*, *loadBinaryFile(...)*, and *saveBinaryFile(...)* methods. There are methods for monitoring file changes, namely *watchFile(...)*, *removeWatch(...)*, and *removeAllWatches(...)*, as well as the *startDrag(...)* method for drag-and-drop operations. Methods for displaying all elements of a file folder (*openFile(...)*) and for deleting files (*moveToBin(...)*) are also available.
- > **Application menus:** You can add menu items to the application window. A menu can contain submenu items. A reaction can be triggered when a menu item is clicked. A previously

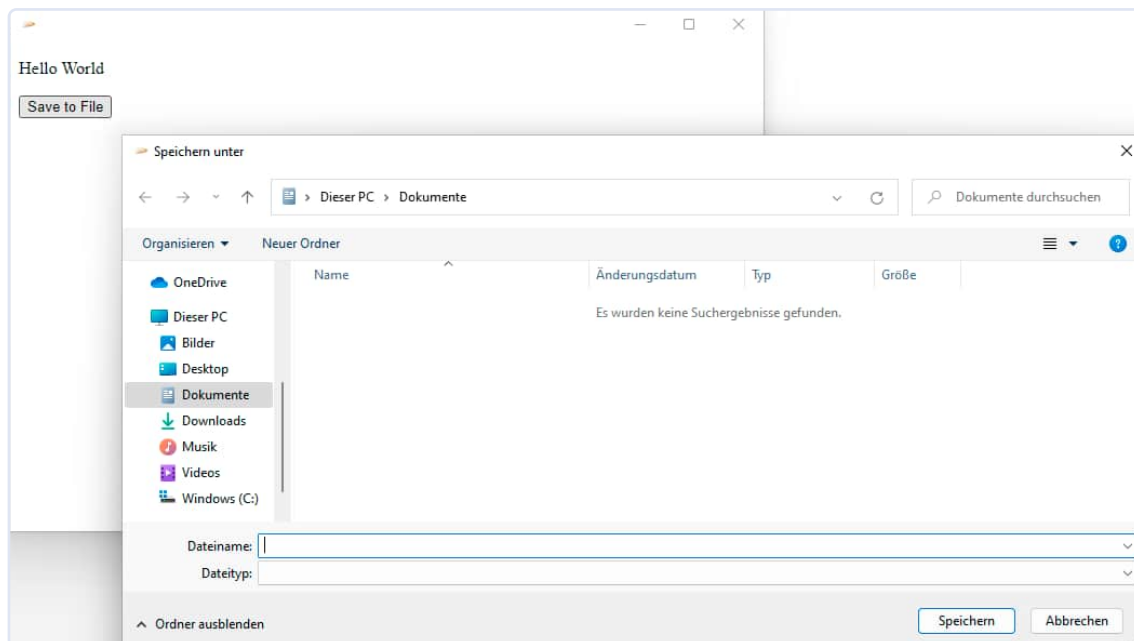


Figure 6: File dialog in Windows — called via the Miletus API.

defined menu is assigned to the application via `application.setMenu(menu)`.

- › **System functions:** For example, you can load an external URL to be displayed in the browser via `shell.openURL(...)`. A system command can be executed via `execute(...)`.
- › **Database access:** The following databases are supported: SQLite, MySQL, MSSQL, PostgreSQL, MS Access (Windows), Firebird, and Interbase. The connection to the database can simply be established via a "connection string" in the following form:

```
let db = new DataBase('sqlite', {database: 'path_to_my/
sqllitedb.db'})
```

- › **Windows Registry:** In Windows, you can access the Registry, i.e. read, write, and check key values.

Further functions of the Miletus API (application window, reading and writing of *ini* files, system messages) are described in the documentation. A special feature is access to the hardware interfaces of the Raspberry Pi. We will look at how this works in the next section.

An App for The Raspberry Pi

We now want to create an app that reads sensors connected to the Raspberry Pi. The app is created as a web app and then packaged into an app package for the Raspberry Pi using Miletus. We use the following test setup:

- › Raspberry Pi version 2 or higher
- › BME280 Environmental Sensor from Bosch Sensortec (see **Table 2** for technical specifications); a breakout board from Waveshare is used in this article
- › Connection of the sensor to the Raspberry Pi via the I²C interface
- › Monitor, mouse, and keyboard connected to the Raspberry Pi

The BME280 measures values for temperature, humidity, and air pressure. We want to read out these values via JavaScript and display them in the app. The sensor is connected directly to the pins

of the Raspberry Pi board. Only four wires are required for this: The BME280 module's VCC (red) and GND (black) are connected to the Raspberry Pi's pin 2 and pin 39, respectively. SDA (blue) and SCL (yellow) are connected to the Raspberry Pi's pin 3 and pin 5. **Figure 7** shows the test setup.

Table 2: Technical specifications of the BME280 sensor [7].

Sensor property	Values
Operating voltage	5 V/3 V
Interface	I ² C/SPI
Temperature range	-40 to 85 °C, resolution 0.01 °C, precision ±1 °C
Air humidity	0 to 100 RH, resolution 0.008% RH, precision ±3% RH, response time 1 s, delay ≤2% RH
Air pressure	300 to 1,100 hPa, resolution 0.18 Pa, precision ±1 hPa
Dimensions	27 mm × 20 mm × 2 mm

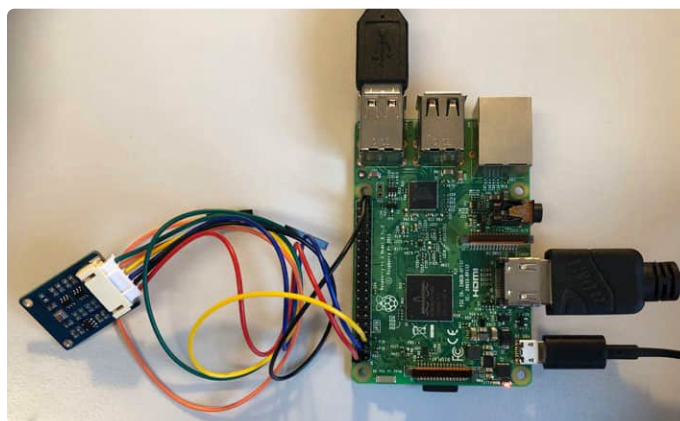


Figure 7: Experimental setup of Raspberry Pi and BME280 sensor board.



Listing 1: index.html of the sensor app.

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <script src="../../js/miletus.js"></script>
    <script src="../../js/bme280.js" defer></script>
    <link rel="stylesheet" type="text/css" href="../../css/style.css" />
  </head>
  <body>
    <h1>Weather Station</h1>
    <div class="container">
      <div class="child">
        
        <p class="label">Temperature</p>
        <div class="value" id="temperature"></div>

      </div>
      <div class="child">
        
        <p class="label">Pressure</p>
        <p class="value" id="pressure"></p>

      </div>
      <div class="child">
        
        <p class="label">Humidity </p>
        <p class="value" id="humidity"></p>

      </div>
      <div class="buttonscontainer">
        <button id="start-reading" class="btn">Start reading</reading></button>
        <button id="stop-reading" disabled class="btn">Stopp reading </button>
      </div>
    </div>
    
  </body>
</html>
```

We install the latest version of Raspberry Pi OS on the Raspberry Pi by writing it to an SD card using a PC. Communication between the two components takes place via the I²C interface, which must be activated in Raspberry Pi OS. To do this, open a terminal on the Raspberry Pi and enter the command:

```
sudo raspi-config
```

In the next step, select the option:

Interfacing Options -> I²C

to activate the I²C kernel driver. After saving the settings and rebooting the Raspberry Pi using:

```
sudo reboot
```

Data can now be exchanged via the I²C interface.

This concludes the preparations. Let's move on to the programming. The project consists of the following files:

- *index.html*: This file is the starting point of the web application (**Listing 1**). The *index.html* file includes the CSS file (*style.css*) and the two JavaScript files *miletus.js* and *bme280.js*.
- *style.css*: The layout and the design of the web application are specified via CSS.
- *miletus.js*: contains the Miletus API.
- *bme280.js*: defines all functions for communication with the sensor.

First, we created a minimal design draft (**Figure 8**). You can use any prototyping tool for this. Alternatively, a hand-drawn sketch will also suffice.

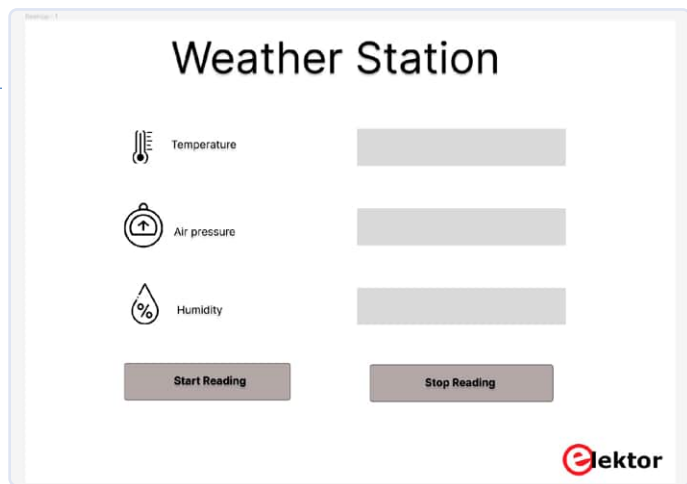


Figure 8: Prototype draft of the app (layout and design).

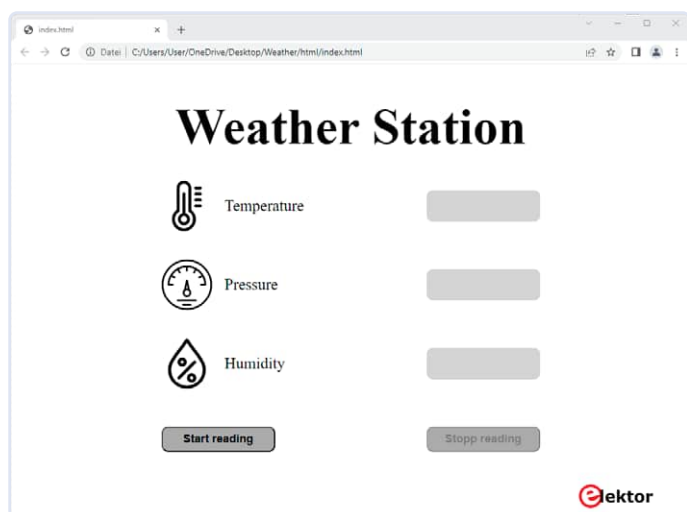


Figure 9: Weather app running in the browser.

In the next step, we then implemented the web app with HTML (structure), CSS (layout, design), and JavaScript (functions).

The app can initially be started again in a browser on any system (Figure 9). This allows you to check out the design of the web app. However, the actual function, i.e. reading out the sensor data, only works on the Raspberry Pi.

Let's take a closer look at the source code. A button labeled *Start reading* is defined in the HTML file:

```
<button class="start" id="start-reading">Start reading</button>
```

When you click on this button, the sensor data is supposed to be read out. The `onClick` event is bound in the `bme280.js` file for this purpose:

```
document.getElementById("start-reading").onclick = startReading;
```

This refers to the `startReading()` JavaScript function in Listing 2.

The sensor data is calibrated, the *Start* button is deactivated, the *Stop* button is activated, the data is read out in a loop, and so on. This process

is determined by the BME280 sensor's conventions and can be found in the documentation and in examples [7] and [8]. For example, if we look at the `readSensorData()` function, we find the following line:

```
let rawValues = await i2c.readBuffer(0xFA, 3)
```

This is used to read out the temperature data. The `i2c` object is defined as follows:

```
i2c = new Miletus.I2C()
```

The Miletus API provides access to the Raspberry Pi's I²C interface. This allows us to read the data from the sensor in the web application using JavaScript. Let's now create the `miletus.config.json` configuration file for the project in Visual Studio Code as described above (command: *Miletus: Initialize config*). Also enter the references to the image files that are to appear in the user interface (e.g. icons).

We can then generate the application package for the Raspberry Pi (command: *Miletus: Package application*). We copy the generated application files to the Raspberry Pi, for example, using a USB stick. The "executable" attribute must be set for the application file. Then, start the app on the Raspberry Pi. It should be executed in a separate window. Click on the button to read out and display the sensor data (Figure 10).

Other Possibilities with the Raspberry Pi

We have looked at communication via the I²C interface. With Miletus, we can also access the following interfaces on the Raspberry Pi [9]:

- **GPIO:** We can read and write data via the 26 GPIO pins.
- **SPI:** This serial bus requires three wires for communication and is based on the master/slave principle.
- **UART:** Among other things, this bus is used for communication with RS-232 or RS-485 interfaces, for example, for data transfer with microcontrollers.

It is also possible to access the *memory buffer*. This is used to read and write the memory buffer at the shell application level.

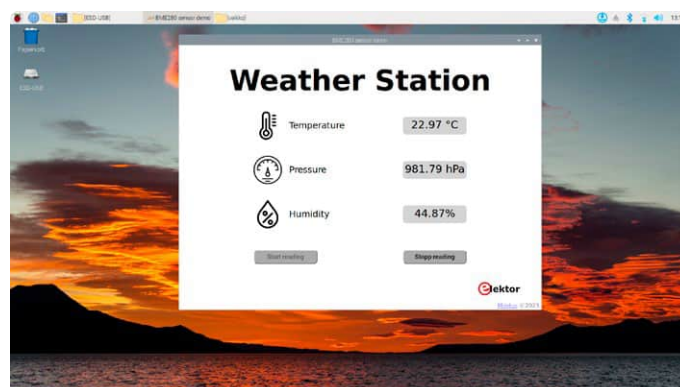


Figure 10: Weather app on the Raspberry Pi.



Listing 2: Function for reading out the sensor data.

```
async function startReading() {
  await readCalibrationData();
  document.getElementById("start-reading").disabled = true;
  document.getElementById("stop-reading").disabled = false;
  doRead = true;
  while (doRead) {
    await readSensorData();
    await new Promise(resolve => setTimeout(resolve, 1000));
  }
}
```



Related Products

- > **Raspberry Pi 4 B (2 GB RAM)**
www.elektor.com/18965
- > **Raspberry Pi 5 (4 GB RAM)**
www.elektor.com/20598

Let's take a look at an example of data exchange via GPIO. Import the *gpio* library using:

```
import { gpio } from 'miletus'
```

Now we can use the `setPin(...)` methods to set the mode of the pin (input or output), use `write(...)` to switch a pin active or inactive (high or low signal), and use `read(...)` to read out a pin. To ensure that the web application's user interface remains responsive during data exchange with the hardware interface, access must be asynchronous. For example:

```
await gpio.setPin(1, 'read')
```

This sets the GPIO port's pin to read mode.

Expandability

The framework can be extended very easily. You can load any library from Miletus and then access the exported functions of this library from the web application. This includes access to *.dll* (Windows), *.dylib* (macOS), and *.so* (Linux/Raspberry Pi) files. In this way, the target systems' special system functions can be used in a web application packaged with Miletus, for example to control hardware connected to the PC via system drivers. Information on this topic can be found at [10].

Reduced Programming Effort

The seemingly unusual step of packaging a web app into a native application offers several advantages. The app can be run on all systems (cross-platform), it behaves like a native application, can be used offline

and has access to the system functions. On the Raspberry Pi, it can send and receive data via the interfaces. This means that a web app can also be used to control electronic circuits. This approach can significantly simplify and reduce the programming effort for many applications. This is especially true if you want to cover several systems simultaneously. ◀

Translated by Jörg Starkmuth — 220616-01



About the Author

Dr. Veikko Krypczyk is a software developer, trainer, and specialist author. He conveys his passion and knowledge in coaching sessions, seminars, training courses, and workshops. You can request workshops and support at <https://larinet.com> and view the agenda in advance.

Questions or Comments?

Do you have questions or comments about this article? Contact Elektor at editor@elektor.com.

WEB LINKS

- [1] Miletus: <https://miletus.org>
- [2] Electron: <https://electronjs.org>
- [3] Visual Studio Code: <https://code.visualstudio.com>
- [4] Miletus: download of the packager: <https://miletus.org/download.html>
- [5] Download of the example code: https://wp.larinet.com/?page_id=663
- [6] Miletus: configuration variables of the Packager: <https://miletus.org/doc/gettingstarted/packager>
- [7] Data sheet of the BME280: https://waveshare.com/w/upload/9/91/BME280_datasheet.pdf
- [8] BME280 driver library: https://github.com/BoschSensortec/BME280_driver/blob/master/bme280.c
- [9] Miletus: GPIO access: <https://miletus.org/doc/reference/gpio>
- [10] Miletus extensions: <https://miletus.org/doc/gettingstarted/extensibility>



From 4G to 5G

Is It Such an Easy Step?

Questions by Roberto Armani (Elektor)

Every day, we're flooded with boasting advertisements for the wonders of the recent 5G concessions. Undoubtedly, they are making an essential contribution to network speed, but is all that glitters really gold? In this interview with Nemo Galletti, an Italian manager of a company specializing in the production of gigahertz antennas for telecom networks, we look at the differences between 5G and the older 4G technology. We also check if perhaps something has been "swept under the rug" by the providers.

Roberto Armani: Nemo, would you mind telling us something about your experiences in telecommunications?

Nemo Galletti (Radio Frequency Systems): 2024 is my 40th year in the telecoms industry. I'm actually working at Radio Frequency Systems, a company controlled by Nokia, and specialize in passive components mainly deployed in the mobile networks, such as special feeders, antennas, waveguides, and radiating cables.

Roberto: How would you describe 5G, in few words?

Nemo: 5G refers to the fifth generation of mobile networks's wireless technology. Compared to previous generation mobile networks, its most significant advantages can be summarized in three key features: speed, capacity and lower latency.

Roberto: In detail, how does 5G technology differ from previous generations (i.e., 4G)?

Nemo: There are relevant technological aspects introduced by 5G networks that differ from previous generations of mobile networks:

- **Data transfer speed:** Compared to 4G, 5G offers significantly faster data transfer speeds. While 4G networks typically provide download speeds of up to several hundred megabits per second, 5G networks can achieve multi-gigabit-per-second speeds. However, rather than "can achieve" we should better say "could achieve," as explained further on.
- **Latency:** Latency refers to the time it takes for data to travel between devices over a network. While 4G networks typically have latency in the range of tens of milliseconds, 5G networks can achieve ultra-low latency in the range of just a few milliseconds.
- **Network capacity:** 5G networks provide significantly higher network capacity than 4G. This means that 5G networks can handle a much larger number of connected devices simultaneously without experiencing degradation in performance.
- **Spectral efficiency:** 5G technology is designed to be more spectrally efficient than the previous technologies. This means that it can transmit more data using the same amount of radio spectrum. 5G achieves this through advanced modulation techniques, more efficient use of available frequency bands, and the use of higher frequency bands.
- **Network slicing:** 5G introduces the concept of network slicing, which allows network operators to create multiple virtual networks within a single physical network infrastructure. Each network slice can be optimized for specific use cases or applications, such as enhanced mobile broadband, ultra-reliable, low-latency communication, and massive IoT deployments.
- **Edge computing:** 5G networks enable edge computing capabilities, bringing computational



Figure 1: Map of the worldwide usage of 5G frequencies. (Source of all pictures, if not otherwise indicated: Radio Frequency Systems)

resources closer to the network edge. This reduces the distance that data needs to travel, improving latency and enabling faster processing for applications that require real-time data analysis or low-latency responses.

- **Improved connectivity:** 5G supports advanced connectivity features, such as beam-forming and Massive MIMO antenna systems. These technologies enhance signal quality, increase network coverage, and improve overall network capacity and performance. Beam-forming allows the network to focus its signal on specific devices, while MIMO utilizes multiple antennas to improve signal quality and capacity.
- **Millimeter wave frequencies:** 5G introduces the use of higher frequency bands, including millimeter wave (mmWave) frequencies. These high-frequency bands provide significantly wider bandwidths, enabling faster data rates.

Roberto: We often associate 5G with deployment of new radio frequencies. Which frequency bands are or will be deployed for 5G networks? Are they different per Country?

Nemo: The 5G standardization bodies defined specific bands to be used by 5G. They are identified with the letter *n* followed by a number. The main *nXX* frequency bands that are or will be allocated to 5G are listed in **Table 1** (but many additional bands can be used). For the most utilized one, *n78*, more precise range values have been indicated in the same table.

The graphic in **Figure 1** is quite a simplification. Europe is not adopting the same frequencies in every country: For example, Germany is not planning to use *n41* for 5G, while the used sub-bands of the most used band, *n78*, will likely differ from country to country. The tendency in Europe is to remain below 3.8 GHz

Table 1: Worldwide band codes and frequency allocation.

Band Code	Frequency (GHz)	Europe	USA/Canada	China	Japan	Korea	Australia
n28	0.7	x	x		x		x
n40	2.3						x
n41	2.5		x	x			
n78	3.5	3.4-3.8	3.7-4.3	3.3-4.9	3.6-4.9	3.4-3.7	3.5
n257	28	x	x	x	x	x	x
n258	26	x	x				x



Figure 2: A typical 4G (5G) tower for 700 MHz and 1,800 MHz bands, with a detailed view of a reflector's interior.

in this band. Since the various frequencies used by 5G are very different, a question could rather be which one is better.

We saw that 5G uses low, medium, and high frequencies. A low frequency (defined as an electromagnetic emission with a frequency of the order of hundreds of MHz) has the ability to penetrate obstacles and reach much further distance than a high frequency, but can carry fewer data per unit of time (typically expressed in bits per second or bit/s). A high frequency (in the order of GHz), on the contrary, has a much lower extension range but has the capacity to transport a lot of data per unit of time. For this reason, an unreliable (i.e. hampered) connection to a high-frequency band can sometimes be slower than a good connection to a low-frequency band.

Therefore, we cannot state that one frequency band is better than another: There are different frequencies for different purposes and the best 5G network is the one that makes good use of all three frequency bands based on the specific need of the moment. A

video stream, access to emails, or a smart thermostat connected to a smart home have different requirements in terms of bandwidth!

Also, for this reason, the new 5G smartphones use a technology called “adaptive beam switching,” which allows them to jump from one frequency band to another to maintain a stable connection. This feature allows for maximizing the benefits of being connected through several frequencies at the same time. This is especially true when we consider millimeter frequencies, the high band n257/n258: Very high frequency allows an extremely high data rate, but even cloudy or foggy weather could have a negative impact on a millimeter-wave beam. Without adaptive beam switching, we would not be able to entrust our connection to millimeter transmission.

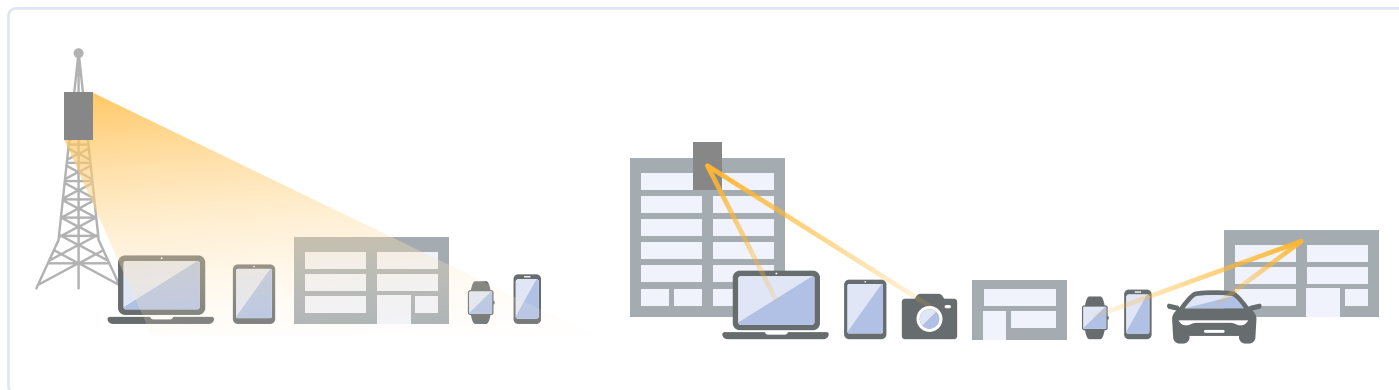
The most-used 5G frequencies, at least initially, will be the medium frequencies, or “sub 6”, between 1 and 6 GHz. Compared to millimeter waves, these frequencies allow for a good data rate, sufficient capacity to penetrate walls, and are less prone to interference. Finally, low frequencies will be ideal for devices, sensors, IoT, controllers, and smart homes, where a low data rate suffices, but high penetration and low latency is required.

Roberto: The topic of frequency also includes 5G antennas. Are they different from 4G ones?

Nemo: In principle, 4G and 5G antennas are not conceptually different. However, some innovative technologies introduced with 5G have an impact on them too. Here, we shall focus on two elements: One is physical, i.e., the size of the dipole shall be as close as possible to half of the wavelength (λ). The simplified formula for calculating the optimal length (in mm) of a $\lambda/4$ dipole is: speed of light [m/s] / frequency [kHz] / 2.

Dipoles can be the same size of the wavelength or a multiple, their shape can be linear, V-shaped, or bent, but their size is always proportional to the wavelength. We saw that the lowest frequency for 5G is 700 MHz, so these dipoles will be longer than those used at 800 MHz by 4G, while for millimeter waves 5G deploys a matrix made of several dipole arrays whose size is minimal. To give an idea, at 700 MHz the typical $\lambda/4$ dipole length is 214 mm, while at 28 GHz we are talking about 5.35 mm.

The second element is related to technology: The “traditional” 4G antenna consists of — for each frequency — a vertical array of dipoles, mounted on a reflector. In **Figure 2**, you can see a typical cell tower with the



▲

set of reflectors mounted in the classic 120° emission scheme, while the picture framed in the bottom-right corner shows the internal structure of a reflector. In this case, inside it, we see the 700 MHz cross-polarized dipoles (golden elements) and, inside them, the smaller, white dipoles for the 1.8 GHz band. From middle band upwards, 5G antennas are often made of a matrix of arrays: several vertical arrays in parallel, initially 6×6 or 8×8, up to 32×32 in Release 15, and higher in future releases. The quantity and sizes of the arrays goes up with frequency, managed in active mode so as to realize Massive MIMO and beam-forming.

The result is that the antennas for low band won't be much different than those we see for 4G, but the typical 5G antennas for higher bands will be like rectangular or squared boxes, getting smaller with the increase in frequency.

Roberto: You've mentioned Massive MIMO, beam-forming, and dipole matrixes. Are these new technologies?

Nemo: Massive MIMO and beam-forming through matrix arrays of dipoles, are not new technologies. However, when we discuss commercial mobile deployment, the first appearance of these technologies was about 20 years ago with the Wimax networks. For several reasons, Wimax networks didn't evolve as expected, so this approach remained confined to specific applications. It is only with 5G that we see the extensive usage of these solutions.

MIMO, (multiple input, multiple output) uses the spatial diversity and spatial multiplexing technique to transmit through different antennas independent and separately encoded data signals — called “streams” — reusing the same time period and frequency resource.

In Multi-User MIMO (MU-MIMO), the transmitter sends different streams to different users simultaneously, using the same time and frequency resources. Spectral efficiency and capacity can be improved by adding additional antennas to support more streams. The MIMO effect relies on the fact that a radio signal between transmitter and receiver is influenced by its environment, with reflections from buildings and

other obstacles resulting in multiple signal paths. The various reflected signals will reach the receiving antenna with differing time delays, attenuation, and direction of travel.

When multiple receive antennas are deployed, each antenna receives a slightly different version of the signal, which can be combined mathematically to improve the quality of the transmitted signal. We talk about “spatial diversity.” This is also achieved by transmitting the radio signal over multiple antennas, with each antenna, in some cases, sending modified versions of the signal.

On top of spatial diversity, spatial multiplexing increases the capacity of the radio link by using the multiple transition paths as additional channels for carrying data. Spatial multiplexing allows multiple, unique, streams of data to be sent between the transmitter and receiver, significantly increasing throughput, and also enabling multiple network users to be supported by a single transmitter.

As you can see in **Figure 3**, beam-forming uses advanced antenna technologies to focus a wireless signal in a specific direction, rather than broadcasting to a wide area.

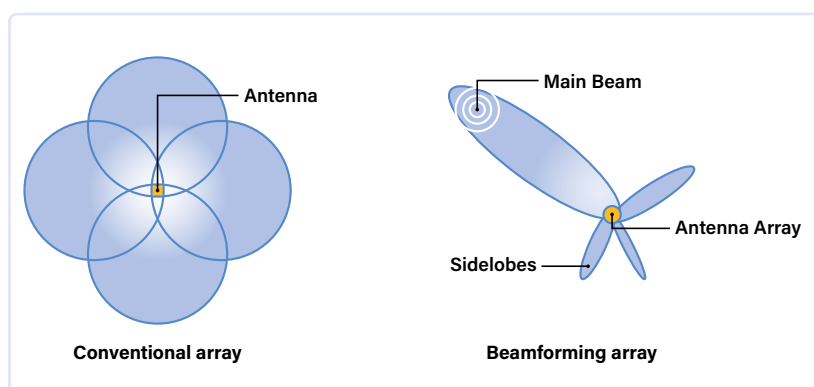
Roberto: Could you explain its operating principle, in more depth?

Nemo: In **Figure 4**, you see the radiation patterns of a standard dipole array antenna (left) and of a

Figure 3: Difference between the standard broadcast footprint of a 4G tower (left) and a modern, optimized 5G one with beam-steering capabilities (right).

Figure 4: Radiation patterns of a standard dipole array antenna (left) and of a 5G array of the latest generation (right) with beam-forming capability.

▼





*If 5G is expected to see the birth of IoT,
6G will embed artificial intelligence in the network,
allowing closer interaction between reality and its
extension in the metaverse.*

5G array of the latest generation (right) with beam-forming capabilities. The beam-forming technology works by focusing the wireless signal in a specific direction, rather than broadcasting to a wide area. This technique reduces interference between beams directed in different directions, enabling the deployment of larger antenna arrays, and therefore is associated with Massive MIMO technology.

3D beam-forming, facilitated by the large number of antennas in a Massive MIMO system, creates both horizontal and vertical beams toward users, increasing data rates on demand, “following” the request coming from a mobile user. Complex algorithms have been developed to coordinate the spatial information obtained from a *channel state information reference signal (CSI-RS)* to enable the base station to communicate with several devices independently and at the same time. The CSI-RS is a signal sent by the base station to the *user element (UE)*, enabling the UE to calculate the *channel state information (CSI)* and report it back to the base station. This information is used by the MIMO system to perform a significant amount of signal processing, using the CSI to represent the channel transfer function in matrix form.

Roberto: How is Europe’s status vs other countries with 5G rollout?

Nemo: From 2016, the EU established a general 5G rollout plan, setting the following milestones:

- First trial networks by the end of 2018.
- Full 5G commercial services in at least one big city by the end of 2020.
- Full 5G coverage in urban areas and coverage continuity on major transport routes (roads, railways) by the end of 2025.
- In March 2021, a further milestone was set: 5G coverage of all inhabited areas by the end of 2030.

By the end of 2020, 23 EU countries reached the target of having at least one big city fully covered by 5G. Only four could not reach this target.

According to the latest surveys, the EU commission forecasts that only 11 countries will reach the end of 2025 with full 5G coverage of all urban areas and major transport routes. In other words, 5G rollout in Europe is proceeding at a slow pace, and the list of the 13 countries that will lag includes also some unexpected names (up to you to identify the “surprises”): Austria, Czechia, Estonia, Germany, Ireland, Poland, Lithuania, Slovenia, Belgium, Bulgaria, Croatia, Cyprus, and Greece.

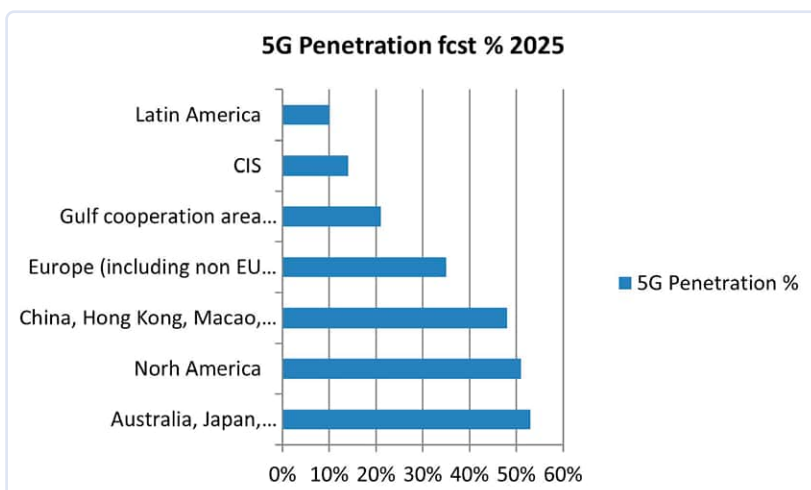
According to the GSM Association forecast, by the end of 2025, the percentage of 5G mobile connections versus total will be as illustrated in the diagram of **Figure 5**.

Good? Definitely not, I would say. Europe will score a miserable 35%. Moreover, it is important to consider that the definition of “5G penetration” in terms of number of 5G connections is a generic concept, since 5G is implemented in several phases (or steps) along a timeline, therefore “number of 5G connections” doesn’t actually mean that the 5G services will really be available — and utilized — on a 5G connection.

Roberto: What are these 5G implementation steps?

Nemo: They are designed by 3GPP, the most important mobile telecom standardization body worldwide. It embodies seven telecommunications standards-development organizations — ARIB, ATIS, CCSA, ETSI, TSDSI, TTA, and TTC — providing the specifications for cellular telecommunications technologies, including radio access, core network, and service capabilities.

Figure 5: Global forecast of 5G penetration by 2025. (Source: GSMA)



3GPP has based the 5G rollout on phases that can coexist and that will usher in the “final” 5G implementation in a time that depends on the speed of the infrastructure investments. Today, each operator in each country is positioned at an intermediate step on this path. Unfortunately, I can say that Europe is generally lagging in respect to other countries, such as the USA, China, and South Korea, which are more advanced with 5G rollout.

About 5G evolution Phases 15, 16, 17, 18, whose standardization has been completed by 3GPP, Europe should have started the rollout of Phase 18 by the end of 2023 (see **Table 2**).

Roberto: So, where is 5G now?

Nemo: In most cases, excluding specific trial zones, 5G is between Phase 15 and Phase 16.

The first step of 5G is called 5G NR (New Radio). This is indeed a complex upgrade, but is only the first step. To simplify, the 5G NR is related to the radio access and all its related protocols and control interfaces. As shown in **Figure 6**, the 5G user element interfaces the 5G Radio Access Network via the NR-Uu interface. The 5G RAN (Radio Access Network) made by gNB's (5G Node B, the baseband unit), has been designed to have its own core control functions, but only a few networks are implementing this control function in a native way.

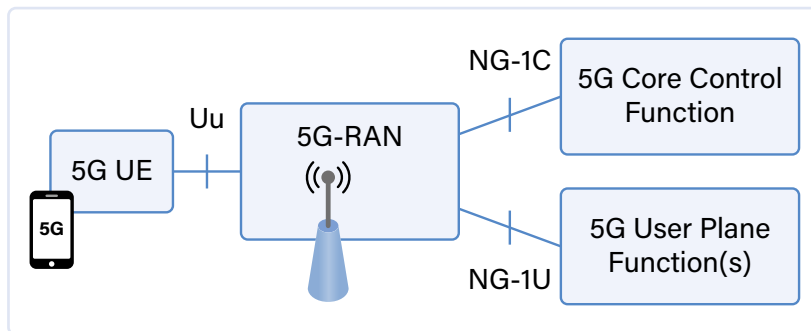


Figure 6: The 5G UE (user element) interfacing with 5G-RAN (Radio Access Network) with its own core (-1C) and user (-1U) functions.

Since the full deployment of a 5G core control function will require time and massive investment, most of the mobile operators are implementing the first phase of the 5G control function on the 4G core control function. We talk about 5G-NSA (non-standalone), to differentiate from the (future) 5G-SA (standalone). In the case of 5G-NSA, the user experience will benefit from only a 15% to 50% improvement in data rate and reduced latency, still far from the promised dramatic improvements reachable by the final 5G-SA.

But: there is a “but.” Even for the 5G-NSA, you need a prerequisite: fully efficient 4G network coverage. However, some countries are far from providing even full 4G coverage in rural areas or even along the main communications stream within these countries (highways, railways).

Roberto: From your words, I understand that the transition is not that smooth, then. Is there perhaps an excess of caution for providers, which is causing delays in the implementation of 5G?

Nemo: 5G is very ambitious and requires huge investments. For private companies, these investments are

Table 2: 5G deployment phases (releases).

Phase (Release)	Year of Implementation or Definition	Purpose	Status (March 2024)
15	2018	Addressing of the New Radio (NR) access concepts	Completed
16	2020	Completion of NR, including unlicensed frequencies and satellite access	Completed
17	2022	Definition of additional services such as enhanced MIMO, spectrum sharing enhancements, coverage enhancements, inclusion of frequency bands up to 71 GHz, enhanced support of private networks, industrial IoT, low-complexity NR devices, edge computing, access traffic steering, switch and splitting support, network automation for 5G, network slicing, advanced V2X service, multiple USIM support and other services	Not completed. Its activation is still ongoing and affected by several delays.
18	2023	Also defined as Advanced 5G. The new specs include edge computing, smart energy and infrastructure, vehicle-mounted relays, evolution of IMS multimedia telephony service, low-power high-accuracy positioning for industrial IoT devices, improvement of network slicing	Definition of Specs completed. Still to be implemented
19	2024	3GPP is actually working on the definition of the features of this Advanced 5G release	Neither fully defined nor implemented.

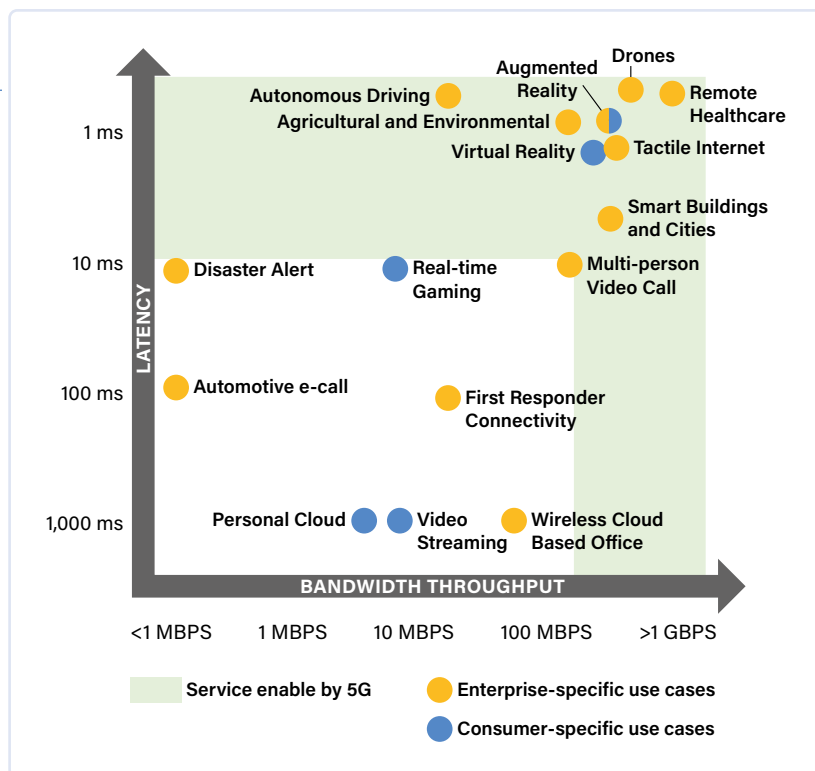


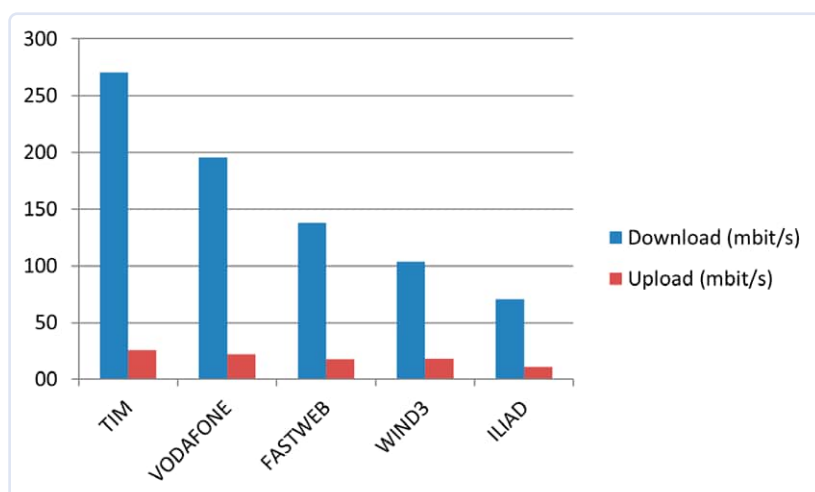
Figure 7: 4G (white) and 5G (light green) application areas of usage, according to the combination of latency and throughput of the connection. (Source: GSMA)

directly connected to expected revenues. And here is the issue, at least in Europe: The ROI (return on investment) is not satisfactory, since in several countries people are happy with the performance of 4G and are not in favor of spending more to get the few-percent-age service improvement offered by 5G-NSA. On the other hand, the expected revenues coming mainly from the corporates and new sectors require services that will be developed only with 5G-SA.

Figure 8: Average download and upload speed of 5G networks in Italy, per network operator, in March 2023. (Source: Opensignal, May 2023)

The diagram in **Figure 7**, derived from GSMA group analysis, shows where the real expectations of 5G in terms of “services enabler” are. The white area is what we can get with an efficient 4G network. The light green area is related to the services that 5G will enable.

Today, we are living in the “chicken-and-egg” paradox: Massive revenues are coming from consumer-specific



use cases, and we can see that, for now, there's not such a massive demand for specific 5G services from this audience. At the same time, a price war ongoing in several European countries (mainly France and Italy) is pushing down the ARPU (average revenue per user). The consequence is that normal consumers are reluctant to pay a premium fee for an increase in features that they don't really need. When we look at typical enterprise (or government) needs, we are looking at services that are not yet implemented by 5G-NSA and that will be available mainly with 5G-SA, requiring further development and investment, requiring funding that several network operators have difficulty obtaining from current revenue levels.

Now, coming back to the initial topic about the difference between 4G and 5G, today I would define the current situation as at least unclear. 4G networks have very different performances around the world, while the performances (data rate per hour) of 5G networks, that are mostly still based on the 4G core, are decreasing as user numbers increase, and these show enormous differences around the world: from 500 Mbit/s in Korea to 300 Mbit/s in China to the current average of around 200 Mbit/s in Italy, but with huge differences between operators, as shown by the graph in **Figure 8**.

In general, 5G “can be” three to five times faster than 4G, or twice as fast as 4.5G in carrier aggregation, but, in attempting to make an effective comparison, the problem is both the reference speed of 4G and that of 5G. The advantages of 5G are actually measured by combining the three factors: speed, latency, and network capacity. Considering them separately doesn't give an idea of the step forward it represents, compared to 4G.

Another example of the status of 5G coverage — updated March 2024 — is shown in **Figure 9** in the 5G coverage map by Vodafone Germany between Hannover, Hamburg, and Berlin (violet is 5G, orange and red are 4G and 4.5G, respectively), illustrates that they are still far from the targets.

Roberto: Should I say, then: “Let's forget about deploying 6G, for now?”

Nemo: Not really. Maybe the picture I depicted looks pessimistic, but I believe that we're just experiencing a temporary slowdown in the 5G rollout. 5G offers — in principle — fantastic features and services. What we need now is to wait for the unavoidable growth of demand of such services in order to have the necessary payload allowing 5G networks to return to profitability.

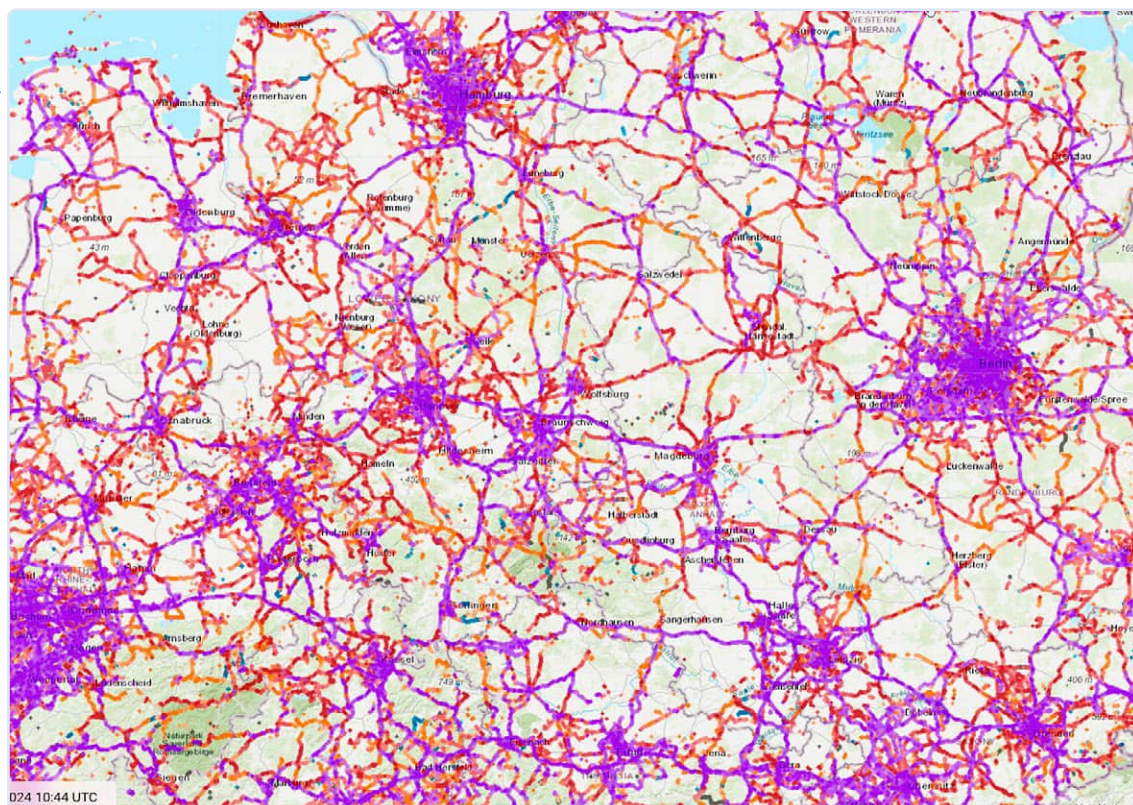


Figure 9: 4G and 5G network deployment in the Hannover–Berlin–Hamburg area (Source: NPERF, nperf.com/en/map/DE/)

The demand for new 5G services did not proceed at the expected pace until now, but is progressing, and will soon exert the required pressure so that 5G can resume its race.

Meanwhile, in December 2023, 3GPP announced that they will start working on the specifications for what will be the 6th Generation, 6G, with the goal of launching the first trials before 2030. So, while it is too soon to predict the exact composition and architecture of 6G, we can still foresee that, since several new features that will be incorporated into the future 6G networks are already the subject of public financing and development within the race of technological supremacy, and therefore could be available shortly before 2030, we will assist in the development of a kind of 5.5G — anticipating and incorporating some of the features designed to be part of 6G, adopting them as a de facto standard.

For example, several announcements associated with the 5G networks, such as the support of a fully automatic driving systems, would probably see a real massive boost only thanks to the better speed, capillary coverage, and integration of different network building blocks that will conceptually be part of the 6G networks.

As of today, the main targets of 6G are a further leap forward on the data rate, up to 1 Tbit/s, a further reduction in latency to the order of 100 microseconds, total coverage of the territory integrating cellular with non-cellular networks including constellations of telecom satellites, new types of hotspots, and new

devices, with the support of AI systems to optimize all the network resources, offering a real personalized service and — this is another important requirement — reducing electromagnetic pollution and energy consumption. If 5G is expected to aid in the IoT explosion, 6G will embed artificial intelligence into the network, allowing closer interaction between reality and its extension in the metaverse. ◀

240233-01



About Nemo Galletti

Nemo Galletti is the Site Leader of RFS Italia, the Italian branch of Radio Frequency Systems GmbH, a multinational company controlled by Nokia at the time of this interview. He studied Electronic Engineering at Politecnico di Milano and gained 39 years of experience in the telecommunications field, initially in R&D for the Alcatel group, then in the overseas commercial sector selling telecommunications networks, before landing at RFS.



About Roberto Armani

Roberto Armani is an electronic engineer. After his studies at Politecnico di Milano, he gained over 35 years of experience in various sectors. Before joining the Elektor team as a Senior Editor, He worked in the computer industry, electronic imaging, telecommunications, material testing equipment, and web publishing. Besides electronics, he loves listening to (and singing) classical music, and taking high-altitude walks in the mountains.

The Elektor Store

Never expensive, always surprising

The Elektor Store developed from the community store for Elektor's own products, such as books, magazines, kits and modules, into a mature web store that offers great value for surprising

electronics. We offer the products that we ourselves are enthusiastic about or that we simply want to try out. If you have a nice suggestion, we are here: sale@elektor.com.



Andonstar AD210 10.1" Digital Microscope



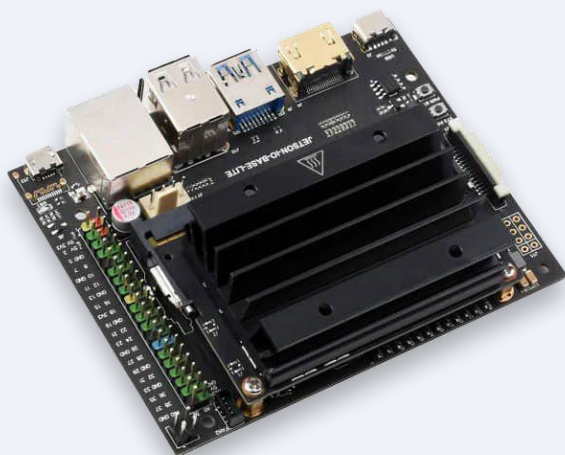
The Andonstar AD210 is a digital microscope with a large 10.1-inch IPS display that offers a 178° viewing angle and supports 1080P video and 12 MP photo capture. With 260x magnification, the microscope provides a clear view of the sides of components on printed circuit boards. The microscope also comes with a 32 GB memory card and a remote control.

Price: ~~€179.95~~

Special Price: €129.95

www.elektor.com/20802

Waveshare Jetson Nano Development Kit Lite



The Waveshare Jetson Nano Development Kit, based on AI computers Jetson Nano (with 16 GB eMMC) and Jetson Xavier NX, provides almost the same IOs, size, and thickness as the Jetson Nano Developer Kit (B01), more convenient for upgrading the core module.

Price: 269.00

Member Price: 242.10

www.elektor.com/20761



UNI-T UT512D Insulation Resistance Tester (2.5 kV)



Price: €289.00

Member Price: €260.10

www.elektor.com/20786

CrowVision 11.6" IPS Capacitive Touch Display (1366 x 768)

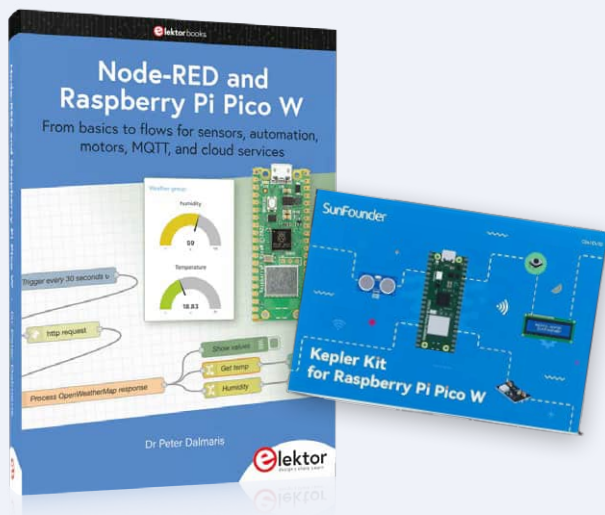


Price: €139.95

Member Price: €125.96

www.elektor.com/20792

Node-Red Development Bundle

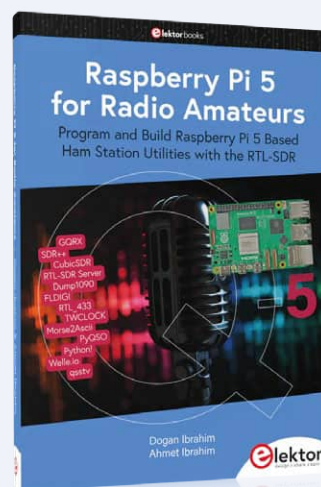


Price: ~~€104.95~~

Special Price: €84.95

www.elektor.com/20849

Raspberry Pi 5 RTL-SDR V4 (Bundle)



Price: ~~€94.95~~

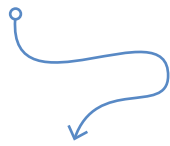
Special Price: €79.95

www.elektor.com/20878



Starting Out in Electronics...

...Balances Out



By Eric Bogers (Elektor)

In this installation, we discuss a few important circuits with operational amplifiers (opamps), including the differential amplifier and the instrumentation amplifier. We will also delve into a topic of paramount importance, especially in audio applications: balanced connections.

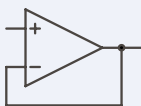


Figure 1: The voltage follower.

The Voltage Follower

A common application of the non-inverting amplifier is the voltage follower (**Figure 1**). When we input “zero” for R_1 and “infinite” for R_2 in the formula for gain, the result is exactly 1. Like the emitter follower discussed some time ago, the voltage follower has a high input resistance and a low output resistance — surpassing the emitter follower by several orders of magnitude in this regard. The voltage follower is also known as an “impedance transformer.”

Bandwidth Limitation

In the non-inverting amplifier, the bandwidth must be limited, and here we also use capacitors for this purpose (**Figure 2**).

Unlike the inverting amplifier, the input resistance can be set independently of the gain, allowing for the use of smaller capacitors. Apart from that, the capacitors are dimensioned in the same way as in the inverting amplifier.

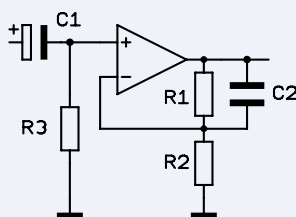


Figure 2: Limitation of the bandwidth.

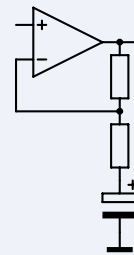


Figure 3: Reduction of DC gain.

An often-encountered variation on this theme involves including a capacitor in series with R_2 , in which case, for DC voltages, the gain is then 1, while the AC gain (beyond the cutoff frequency) is determined, as usual, by R_1 and R_2 (**Figure 3**).

The Summing Amplifier

A variation on the “inverting amplifier” theme is the summing amplifier (**Figure 4**), which sums the voltages of multiple inputs.

Usually, equal values are used for the input resistances, but the circuit also functions

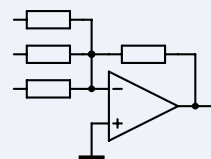


Figure 4: The summing amplifier.

with unequal input resistances — the individual voltages to be added must then be multiplied by the respective factor.

Summing amplifiers are commonly used in mixing panels to sum the individual channels — not only for the master or subgroup but also in the aux signal paths. Since the node (summing point) is a virtual ground, the individual channels can be mixed freely without the risk of interference. The low impedance of the summing point also makes the sum signal connection relatively insensitive to interference.

Naturally, this connection (“rail”) does not carry exactly a zero potential, and its impedance is also not zero — therefore, its insensitivity to interference is limited. To further minimize interference, the individual signal lines are preferably placed between two ground connections. It is also possible to make the signal symmetrical, but this naturally requires two bus connections.

Symmetrical Connections

Before we examine the differential amplifier, we briefly enter an important side path: To understand its great importance, we must first pay attention to the differences between symmetrical and asymmetrical connections.

As we saw at the beginning of this series of articles, we need an outward and return line to allow an electric current to flow. In DC technology, we speak of a positive and negative line, and in AC technology, we speak of phase and ground (or zero).

For connecting speakers, an (unshielded) two-wire cable with a twisted core is often used, where one wire is the phase and the other is the ground. In principle, it does not matter which wire is used for this; the connection of the cable to the connector is standardized, and one of the wires is marked.

Speaker cables are not particularly critical for picking up interference signals: they typically carry a high signal level and are terminated with a very low impedance.

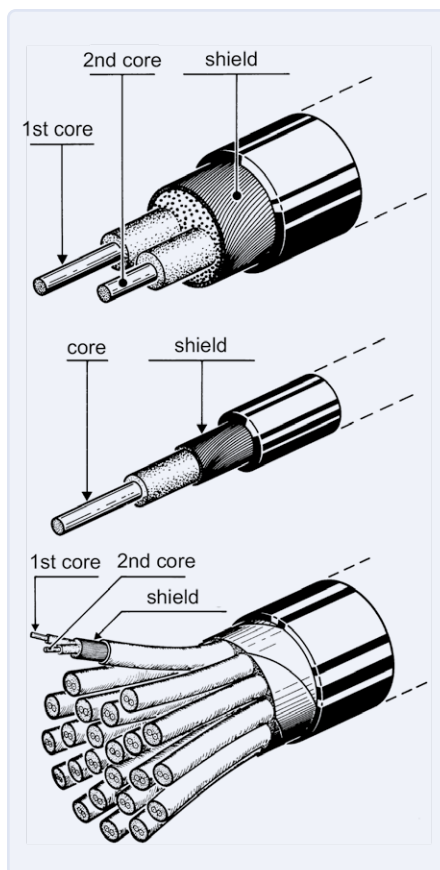


Figure 5: Symmetrical and asymmetrical lines and a multicore cable (bottom).

In the simplest case, such a shielded cable consists of an inner conductor with a shield around it; the inner conductor carries the signal while the shield is connected to the ground. For cables which are not too long and have a high signal level, such an unbalanced connection is entirely sufficient; in practice, it is often used in this way. All cables equipped with RCA or Cinch connectors or mono jack connectors are of this unbalanced type. (Stereo jack connectors can carry a balanced mono signal but can also be used in other ways.)

In Figure 5, we see a typical example of an asymmetrical cable in the center.

A symmetrical cable is always used for long cables and/or at low signal levels. Here, the signal is transmitted through one conductor in the correct phase and through the other conductor in the opposite phase, with both conductors within a common shielding. At the top of Figure 5, we see an example of such a cable. But, what is the actual advantage of this “double” signal transport?

Take a look at **Figure 6**. Figure 6a shows an example of an asymmetrical connection between two devices. If interference occurs,

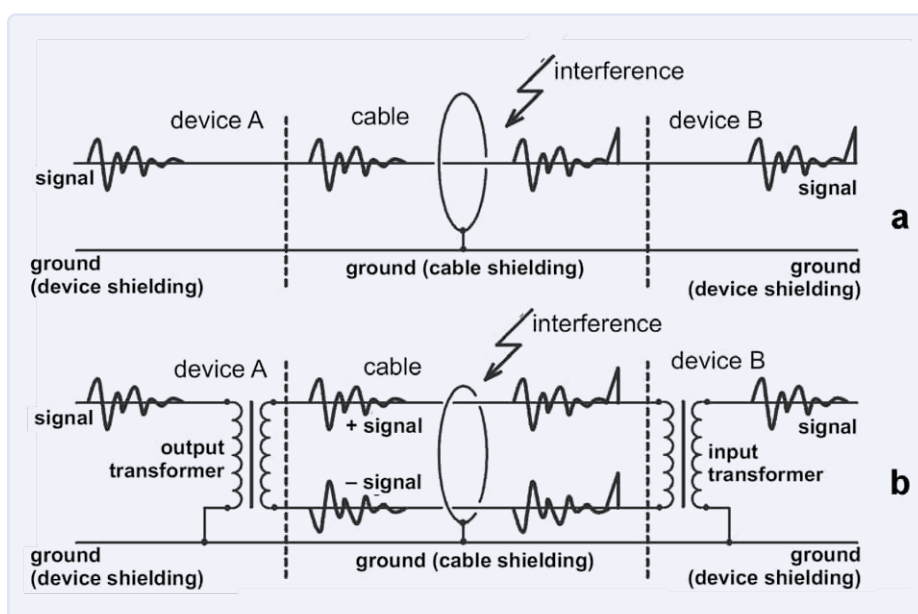


Figure 6: Asymmetric (a) and symmetric (b) signal transmission.

it is indeed diverted to ground by the shielding, but a small part of it still penetrates the signal conductor. (For the following discussion, it is not relevant whether this is because the shielding is not perfect or because the interference enters through the ohmic resistance of the shielding. We will assume here that the shielding is not 100% effective because that is a “demonstrative” representation of things.)

In a balanced connection (Figure 6b), one conductor carries the signal in the correct phase, while the other carries the same signal, but in the opposite phase. Similarly, the interference signal can penetrate the signal-carrying conductors, but this interference signal (unlike the “useful” signal) has the same phase on both conductors. In the receiving device, the signals on both conductors are subtracted from each other (thus forming a *differential* signal); because the interference signal has the same phase on both conductors, it is completely cancelled out (at least in theory).

You might wonder why we still need shielding if, in a symmetrical connection, interference signals are completely suppressed. Well, firstly, the two signal-carrying conductors never pick up the interference signal identically — even when the conductors are twisted. Secondly, the formation of the differential signal is never entirely perfect.

Differential input stages equipped with 1% resistors achieve a common-mode rejection of about 40 dB; professional signal transformers (depending on the frequency) achieve about 60 dB. In practice, this is often not enough. However, when we combine balanced signal transmission with proper shielding, the result is an excellent signal:noise ratio.

The Differential Amplifier

The differential amplifier forms the difference of two input signals (Figure 7). The gain is calculated as for the inverting amplifier; for the output voltage, thus:

$$U_{\text{out}} = (U_{\text{in+}} - U_{\text{in-}}) \cdot \frac{R_2}{R_1}$$

One of the issues of the differential amplifier is the extremely unequal input impedance. This problem can be circumvented by not equaling the value of the resistors but keeping their ratio equal. In practice, however, this again has a detrimental effect on *common mode* suppression (suppression or attenuation of same-phase signals): if we use equal resistor values, in many cases, they will come from the same production series — meaning that although they will differ slightly from the printed value, they will be virtually the same among themselves.

When using different resistor values, the two R_2/R_1 ratios will never be completely equal, meaning that an offset signal will be amplified along with it.

When using 1% metal film resistors, we usually end up with a maximum common mode suppression of 40 dB.

The Instrumentation Amplifier

The greater the gain delivered by a differential amplifier, the greater the issues with uneven input impedances. For amplifications of 20 dB or more — that is, 10 times or more — you can consider using so-called instrumentation amplifiers (Figure 8). The first advantage of this setup is that the two inputs have an equal input impedance, which is usually set to a usable (lower) value by means of a resistor to ground.

Please note the special connection of the resistors at the input amplifier: When applying symmetrical signals, there is a virtual ground point in the middle of R_4 , so, in the formula for the gain, half (!) of the value of R_4 must be entered. On the other hand, for common mode signals, R_4 may be considered not present, so the gain of common mode signal components is equal to one.

The increase in asymmetrical damping is in the order of magnitude of the gain of the input stage. Since our primary concern is to maximize asymmetrical damping, in most cases, we will assign R_1 and R_2 the same value — say, 10 k Ω — and let the input stage provide the gain. If we set it to 20 dB, a common mode attenuation of 60 dB is achievable for the circuit as a whole.

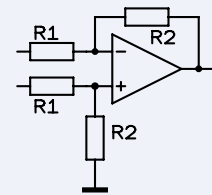


Figure 7: The differential amplifier.

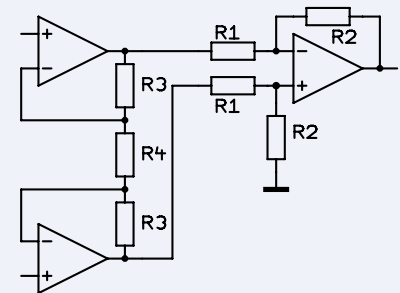


Figure 8: Instrumentation amplifier.

We’ll leave it here for this time; the next installment will cover many interesting practical circuits with opamps. ◀

Translated by Hans Adams — 240127-01

Editor’s note: This series of articles, *Starting Out in Electronics*, is based on the book, *Basiskurs Elektronik*, by Michael Ebner, which was published in German and Dutch by Elektor.

Questions or Comments?

Do you have questions or comments about this article? Feel free to contact the Elektor editorial team at editor@elektor.com.



Related Products

> **B. Kainka, *Basic Electronics for Beginners* (Elektor, 2020)**
Book: www.elektor.com/19212
Ebook: www.elektor.com/19213

PROTEUS DESIGN SUITE



Driving forward with Manual Routing

Push and Shove Routing
for dense layouts

Dedicated Differential
Pairs Routing mode

Length Matching and
Net Tuning Support

Visual DRC shows legal
paths for route placement

HEAVY
TRAFFIC

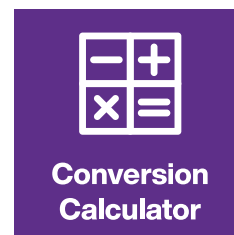
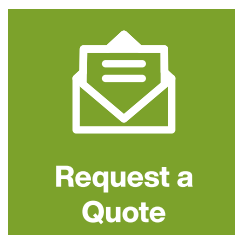
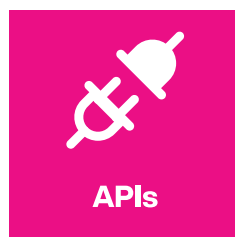
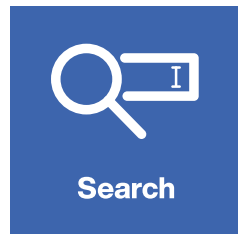
FASTER
ROUTING
AVAILABLE

labcenter
Electronics

www.labcenter.com

info@labcenter.com





Ordering made easy

Tools to search, check stock and purchase

mouser.com/servicesandtools

