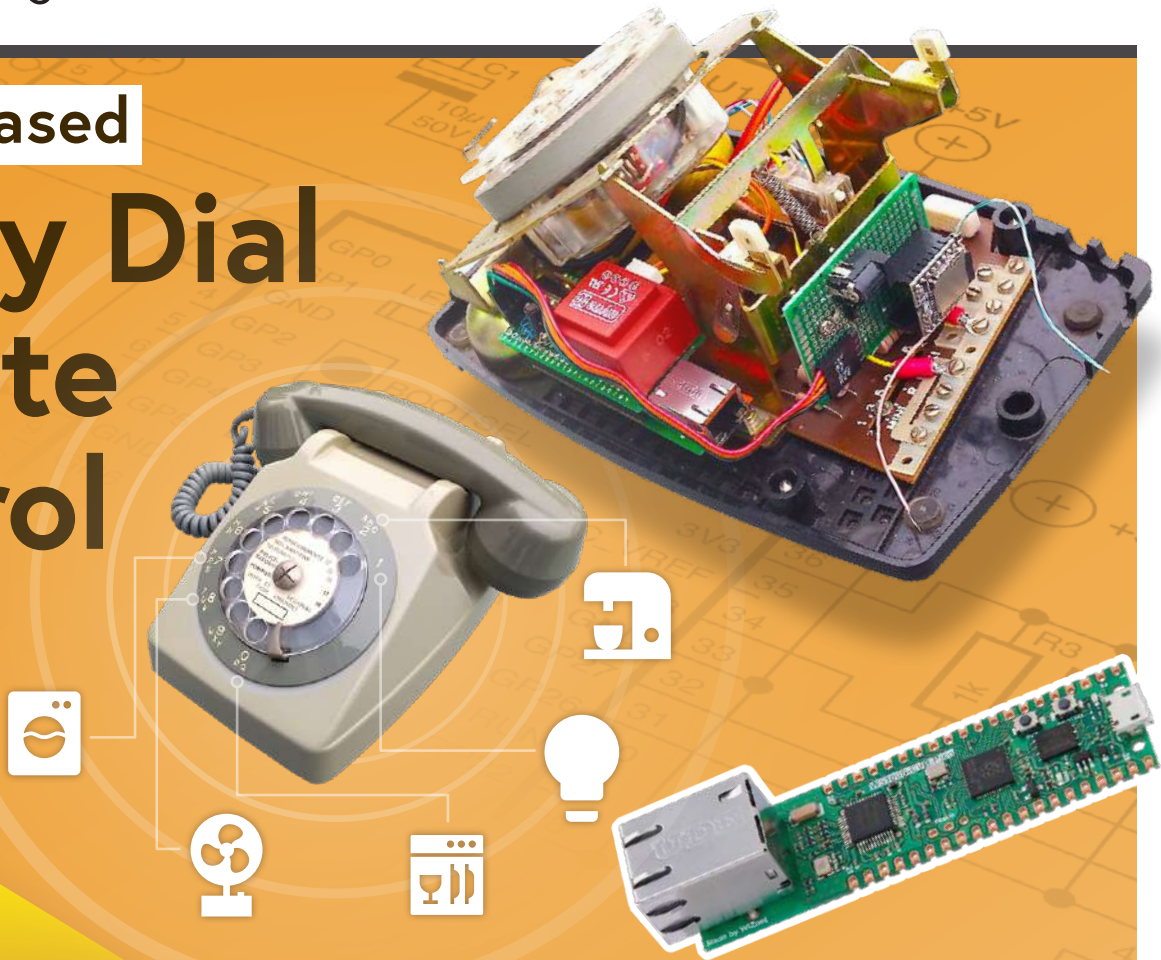


RP2040-Based

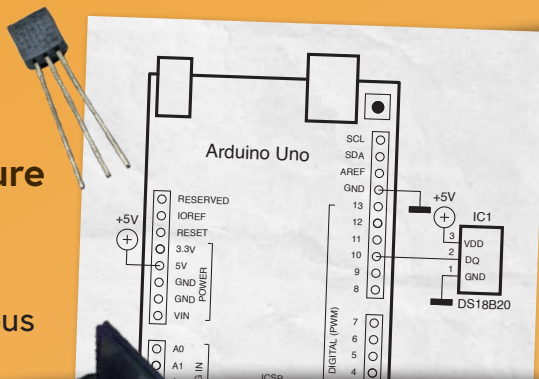
Rotary Dial Remote Control



FOCUS ON

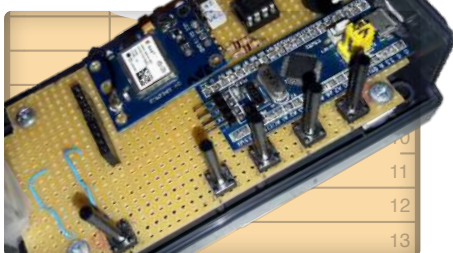
IoT & Sensors

DS18B20 Temperature Sensor
Measure on the 1-Wire bus



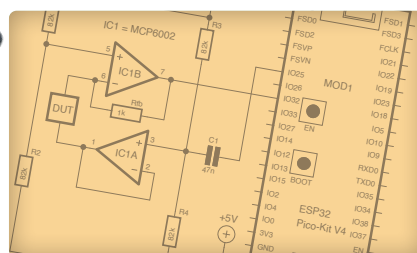
matter

The Matter Standard
Will it save the smart home?



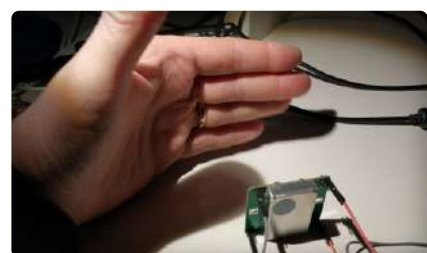
GPS-Based Speed Monitor
Avoid speeding tickets

p. 11



ESP32-Based Impedance Analyzer
A simple, low-cost solution!

p. 30



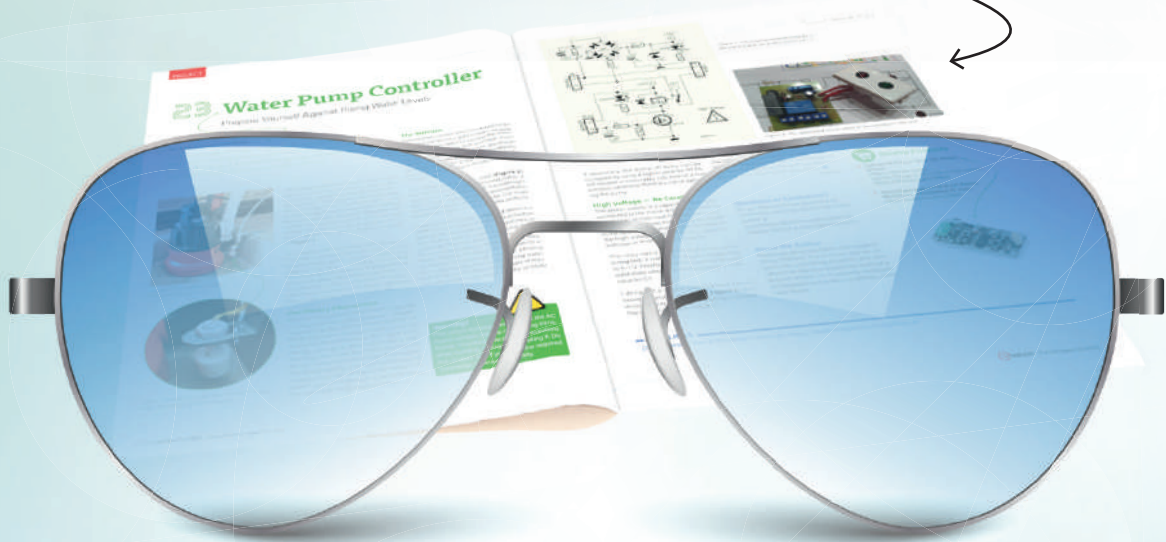
Doppler Motion Sensor
Use microwaves to detect movement

p. 90



GET READY
FOR AN

Epic Summer of Projects!



Calling all electronics enthusiasts and electronics makers! The Elektor Magazine Circuit Special 2023 is coming up in August and it's packed with numerous remarkable projects. Get ready to dive deep into the captivating world of projects and **UNLEASH YOUR INNER CREATIVITY!**

STAY TUNED — all of our members will receive this special issue and it'll be in the Elektor Store!



www.elektormagazine.com/circuit-special

Volume 49, No. 522
July & August 2023
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. 2023

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



AI Horror, AI Fun

What's the opposite of Marlon Brando? Don't know? Then look at the (creepy) YouTube video that our author Ilse Joostens links to in her article on page 46. This time, the column is about Artificial Intelligence, and especially about the countless tools that generate images, text, and program code. By the way, AI is now creating entire videos, and here, too, the results are amazing, sometimes funny, but often a bit frightening. One might wonder why artificial intelligence tends to generate horror and apocalyptic images so often — even in innocent 1990s-style pizza commercials (www.youtube.com/watch?v=MpvEXrhnoyW).

However, we natural intelligences are still masters of the situation. And as long as that is the case, we should make the most of AI. Programmers and electronics engineers in particular have a lot of options here, which is why we at Elektor have begun exploring. I am quite sure that you will soon be able to search for a suitable article from our huge magazine archive with the help of AI. Manually, it would take years to track down the countless cross-connections between the projects and basic articles and to transfer the whole thing into a complete body of knowledge (if we could even think of a workable system for this).

We will also deal with the automatic generation of program code in the coming issues. I have friends who are now throwing ChatGPT on as a matter of course to have batch files written for editing audio files. I am curious to see what else is in store here, especially for electronics engineers. Maybe you have already made your own experiments? Have you had good or bad experiences? Write me at editor@elektor.com!



Submit to Elektor!

Your electronics expertise is welcome! Want to submit an article proposal, an electronics tutorial on video, or an idea for a book? Check out Elektor's Author's Guide and Submissions page:

www.elektormagazine.com/submissions



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

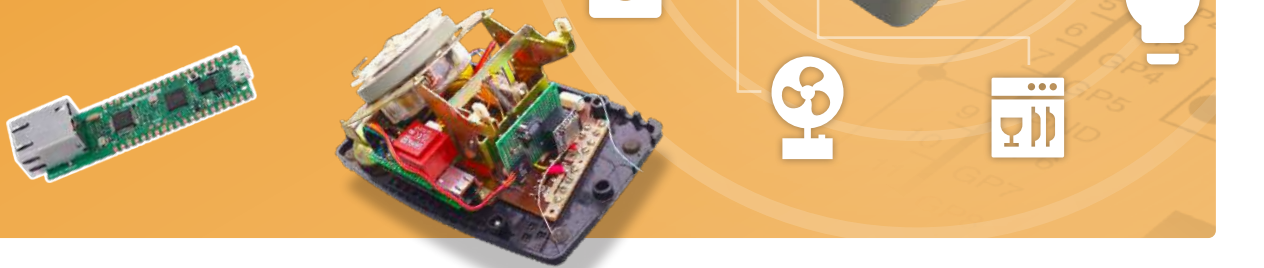
The Team

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, Hedwig Hennekens, Alina Neacsu, Dr. Thomas Scherer, Clemens Valens, Brian Tristram Williams | **Regular Contributors:** David Ashton, Tam Hanna, Priscilla Haring-Kuipers, 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

Rotary Dial Phone as Remote Control

To Switch on the Lights, Dial 1;
For the Coffee Maker, Dial 2

6



Regulars

- 3 Colophon**
- 23 Starting Out in Electronics...**
Follow the Emitter
- 26 Developer's Zone**
Arbitrary, Independent Hysteresis Levels for Comparators
- 38 HomeLab Tours**
Encouraging DIY
- 46 From Life's Experience**
Modern Luddism
- 110 Peculiar Parts**
Microprocessors for Embedded Systems
- 124 Retronics**
Transverter for the 70 cm Band
- 126 Ethics in Action**
Climate Calling Engineers
- 130 Hexadoku**
The Original Elektorized Sudoku

Features

- 40 The MCCAB Arduino Nano Training Board**
All-in-One Hardware for the "Microcontrollers Hands-On Course"
- FOCUS**
- 48 Sensor 101: The DS18B20 Temperature Sensor**
Connection to the 1-Wire Bus

FOCUS

- 88 Build a Cool IoT Display**
With the Phambili Newt
- 97 A Bare-Metal Programming Guide (Part 1)**
For STM32 and Other Controllers
- 106 Review: Siglent SDM3045X Multimeter**
- 112 Microcontroller Documentation Explained (Part 3)**
Block Diagrams and More

Industry

FOCUS

- 54 Is Matter the Thread to Save the Smart Home?**
New Standards to Simplify the Smart Home

FOCUS

- 58 A Matter of Collaboration**
Developing with the Thing Plus Matter Board and Simplicity Studio

FOCUS

- 62 Infographics: IoT and Sensors**

FOCUS

- 64 Matter, ExpressLink, Rainmaker — What Is This All About?**
Q&A with Espressif's Amey Inamdar

FOCUS

- 68 Selecting Microcontroller Dev Kits for IoT and IIoT Applications**
An Introductory Guide
- 74 Capacitors Do Not Always Behave Capacitively!**



Build a Cool IoT Display With the Phambili Newt

88



Low-Power LoRa Weather Station

Build a Long-Range
Weather Station
by Yourself

116

Projects

FOCUS

- 6 Rotary Dial Phone as Remote Control**
To Switch on the Lights, Dial 1; For the Coffee Maker, Dial 2

FOCUS

- 11 GPS-Based Speed Monitor**
No More Speeding Tickets

- 16 RGB Stroboscope with Arduino**
A Colorful Adaptation of a Useful Instrument

FOCUS

- 20 Wireless Emergency Push Button**
Enhanced Safety with LoRa

- 30 ESP32-Based Impedance Analyzer**
Simple, Low-Part-Count, and Inexpensive!

- 78 An NTP Clock with CircuitPython**
Why Should You Use This Programming Language?

FOCUS

- 90 The HB100 Doppler Motion Sensor**
Theory and Practice

FOCUS

- 116 Low-Power LoRa Weather Station**
Build a long-range weather station by yourself

Next Edition

Elektor Magazine Circuit Special Edition (August & September 2023)

In the tradition of the Summer Circuits Guide, next edition will be extra thick, filled with more than 50 DIY-projects, retro circuits, tips and tricks and much more!

From the contents:

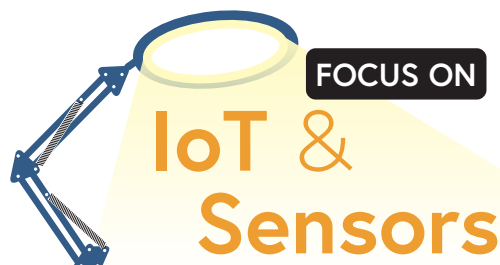
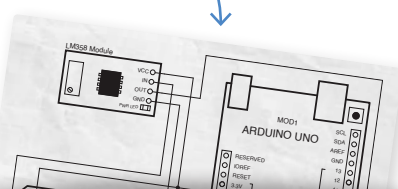
- > Active rectifier
- > A Low-Cost Frequency Standard
- > Simple Dynamic Compressor
- > Tiny Solar Supply
- > THD Generator
- > Programmable Video DAC
- > Large RGB Digit
- > ChatGPT and Arduino
- > Solar-Powered Christmas FM Radio Ball
- > Tiny DCF77 Simulator

And much more!

Elektor Magazine Circuit Special 2023 edition will be published around **August 9, 2023**. Arrival of printed copies with Elektor Gold Members is subject to transport. Contents and article titles subject to change.

The HB100 Doppler Motion Sensor Theory and Practice

90



Rotary Dial Phone as Remote Control

By Clemens Valens (Elektor)

Even when lights and appliances are controlled by a home automation system, it is often desirable to have an override option to switch on or off a lamp, a ventilator or some other device. The modified rotary-dial phone presented in this article allows this by dialing the number of the appliance. At the same time, it forms a now sought-after decorative object.



In this project, we turn an old analog phone with rotary-dial into a remote control and alarm for a home automation system. Besides as a (decorative) remote control, the modified phone can also be used as a prop in, for instance, an escape game. I am sure many other applications can be imagined too.

RP2040-Based Design

The brain for this project is an RP2040 microcontroller mounted on a WIZnet W5100S-EVB-Pico board (**Figure 1**). This is basically a Raspberry Pi Pico board with a W5100S "Internet" chip added to it (and an Ethernet connector). Therefore, it is pin-compatible with the popular Pico board. The only thing to keep in mind is that some of its pins (GPIO16 to GPIO21) are used for talking to the W5100S.

Wired vs Wireless

Even though today wireless networking appears to be the standard, there are still many applications for cabled

or tethered Ethernet. One advantage of wired Ethernet is that it is possible to provide power to the connected nodes. This feature is used in this project (see below). Furthermore, as rotary-dial phones have always had a cable running to a wall outlet somewhere, it would even be strange to have one without. Therefore, the W5100S-EVB-Pico board is a defensible choice for this application.

Remote Control Over MQTT

The home automation controller (HAC) controlled by the rotary dial phone is Home Assistant (HA), a popular automation platform that is gaining traction every day. However, since the remote control talks MQTT, it can easily be integrated in other home automation systems too.

Connecting the MCU to the Phone

The phone I used, a classic French model (S63), produces pulses at a rate of 10 Hz. The vintage phone's dialing mechanism (**Figure 2**) can be viewed as two switches:

one indicates that dialing is in progress (busy/idle) while the other closes a certain number of times depending on the chosen digit. A '1' produces one pulse, a '9' nine pulses and '0' is ten pulses.

Connecting this mechanism to the W5100S-EVB-Pico board is easy; connecting it without modifying the phone requires a bit more thought. I wanted to keep the phone as close to its original state as possible. By choosing the right connection points in the phone and adapting the required pull-up resistors, this can be achieved (Figure 3).

The handset rests on a normally open (NO) switch, so lifting the handset closes it. This changes the electrical circuit of the phone and with it the impedance of some

of the connection points of the dial mechanism. Fortunately, this can be handled by choosing relatively low values for some of the pull-up resistors.

High-Voltage Bell Inverter

The bell of the phone (Figure 4) needs a relatively high AC voltage to ring, at least 35 V_{AC} as I found. I therefore built a simple low-power voltage inverter controlled by the MCU. The MCU generates two 50 Hz square waves with a 180° phase difference that drive the secondaries of a small 230 V_{AC} mains transformer. The bell is connected to the primary side of the transformer. A small 2x 9 V, 1 VA transformer suffices to get a decent bell volume. The complete schematic of the Phone-to-Microcontroller interface is shown in Figure 5.

Figure 2: The dialing mechanism of the S63 phone produces pulses at a rate of 10 Hz.

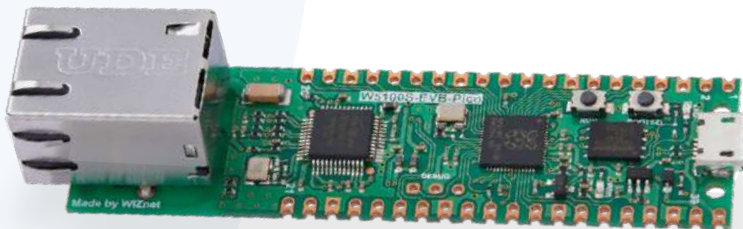


Figure 1: The WIZnet W5100S-EVB-Pico board has an RP2040 microcontroller connected to a W5100S Internet-over-Ethernet IC. The board has the same pinout as the Raspberry Pi Pico board. However, note that GPIO16 to GPIO21 are also connected to the W5100S chip. (Source: WIZnet)

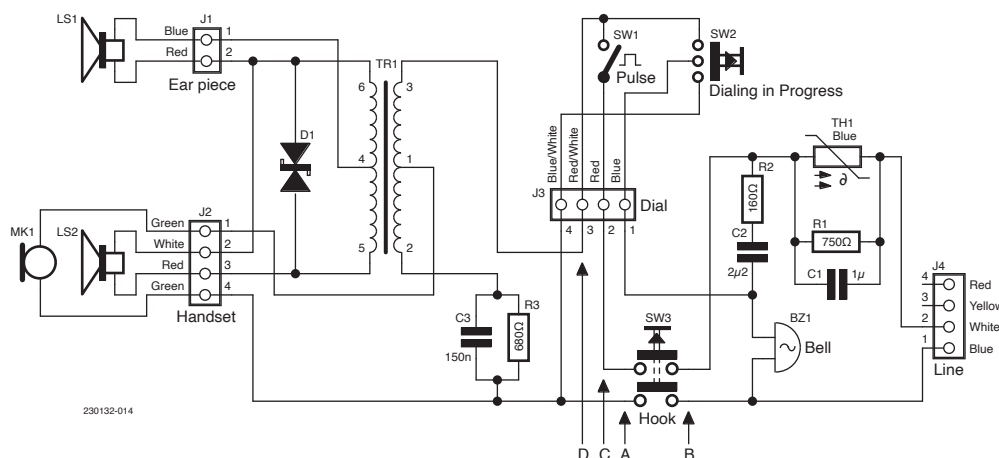


Figure 3: To connect the microcontroller board to the phone without modifying the phone, I connected it at the points A (GND), B (hook lifted), C (dial pulses) and D (dialing in progress).

Handler (see the file *mqtt_interface.h*). Once I added it to my millisecond timer callback, MQTT worked fine.

mDNS

Because my Home Assistant installation relies on DHCP to connect to the network, it may be attributed a new IP address at any moment. To find its peripherals and to communicate with them, HA uses multicast DNS, a.k.a. mDNS. As I did not want to hard-code a temporary IP address in my program and recompile it every time the HAC's address changes, I added mDNS support to it. For this, I adapted the WIZnet DNS library, as mDNS is quite similar to plain DNS. Now, the program issues an mDNS request to obtain the HAC's IP address before trying to connect to it. This makes the system much more flexible and reliable.

Handset Detection and Dialing

Reading the switches of the dialing mechanism requires debouncing. This is implemented in software. Once properly debounced, counting the pulses becomes a trivial task.

Dialing with the handset on the phone generates single-digit messages; dialing with the handset lifted produces a multi-digit number, which is sent after putting down the handset. The digit or number is sent as the payload of an MQTT packet. Note that digits are sent 'minus one' to make the 10 values fit inside one character. So, 1 is sent as 0, 9 as 8, and 0 is sent as 9.

Texting

When the handset is lifted, dialing produces besides a multi-digit number also letters to compose text messages with. The 26 letters of the alphabet are distributed over the digits 2 to 9, three characters per digit, except 6, which only has two ('m' and 'n'). Digit 0 also has two ('o' and 'q'), digit 1 has none. This is the default distribution printed on the French S63 phone (**Figure 8**). For some reason, there is no 'z', and so I added it to digit 0.

To select a letter, the corresponding digit must be dialed up to three times. As an example: 'a' is 2, 'b' is 22 and 'c' is 222. To write 'aaa' the dial sequence is 2-2-2, for 'abc' it is 2-22-222. Here, the dash represents a pause of at least one second. A delay of less than one second between two identical digits selects the next letter attributed to the digit. This method is very similar to that used on mobile phones from 2000 to compose text messages, except that there is no display. You must keep track of your message in your mind (good exercise!).

When the handset is placed back on the phone, the composed message is sent, together with the multi-digit number, to the HAC as the payload of an MQTT packet.



Figure 8: Texting with this old phone is easy thanks to the character map printed around the dial. As an example, "elektor" is dialed as 33-555-33-55-8-0-77, where the dash represents a pause of one second.

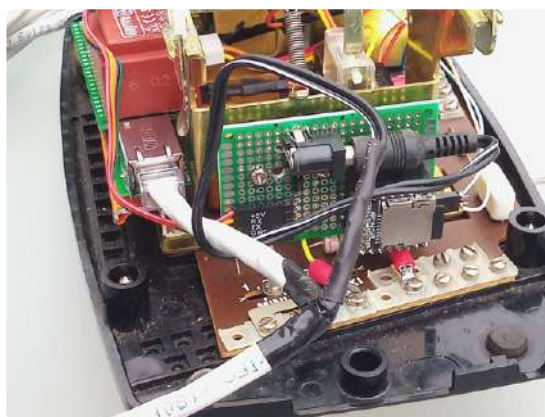


Figure 7: Power over Ethernet done the inefficient way. It works as long as the input voltage is high enough to overcome the voltage drop introduced by the cable.

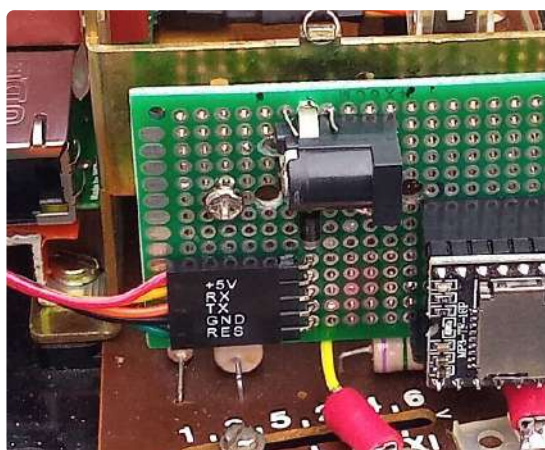


Figure 6: The MP3 player module and power supply connector share their own little board.

The HAC can then forward it to, for example, a texting service or display it on its dashboard.

Some Things You Might Want To Change

The interface presented in this article was designed to work with a French S63-type phone and built to fit inside it (**Figure 9**). You may have to modify some parts for use with another model.

Instead of Ethernet, you may prefer Wi-Fi or another wireless communication method. This is, of course, possible.

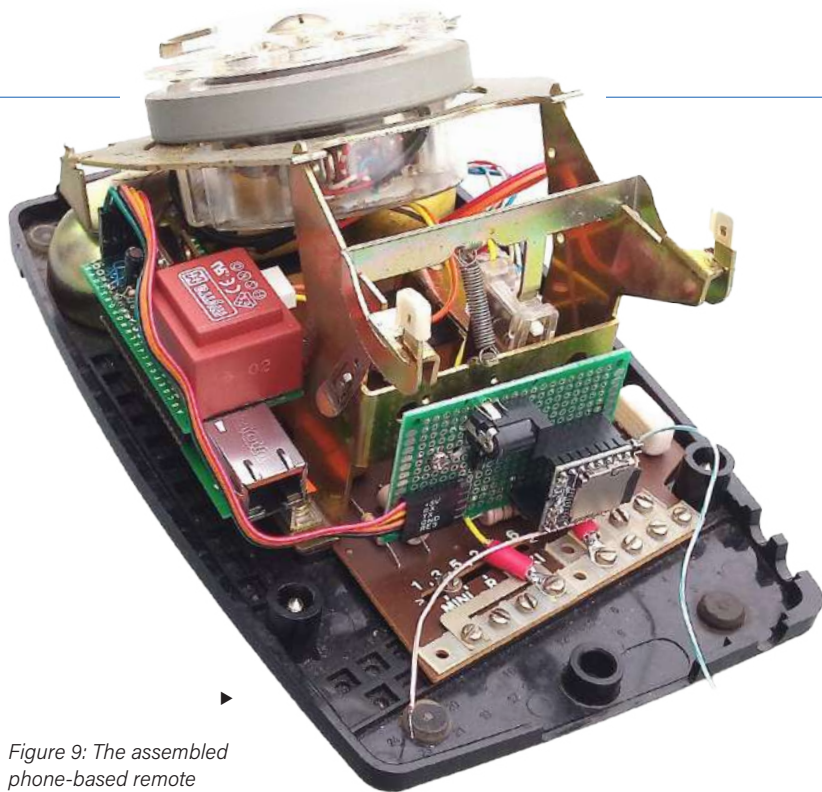


Figure 9: The assembled phone-based remote control. On the left, you can see the transformer of the high-voltage bell inverter plugged on top of the W5100S-EVB-Pico board. A cable connects this assembly to the MP3 player module and power connector on the right. The connections to the dialing mechanism are not visible, as they are on the other side of the phone.

The software can easily be adapted to other physical layers, as the application program uses a standard network driver API. The TCP/IP stack is handled in its entirety by the W5100S IC, and therefore it can be replaced by another communication module.

Adding a good battery would allow for an untethered remote control (that must be recharged every once in a while). If the phone's original circuits may be removed, then there is plenty of space for one.

The high-voltage inverter for the bell can be built in different ways. I used a transformer because I had one at hand, but a suitable low-cost converter module found online may do the job as well.

VoIP

Currently, the microphone of the phone is not used, but one could imagine using it for a voice-over-internet or VoIP service. The RP2040 supports I²S, so standard microphone interface ICs can be used for this.

The remote control application consists of an Arduino sketch and an easy to install Arduino library based on the official WIZnet repository. It can be found at [1].

230132-01

Questions or Comments?

Do you have technical questions or comments about his article? Email the author at clemens.valens@elektor.com or contact Elektor at editor@elektor.com.



Remote Control Functions

The modified rotary-dial phone has the following functions:

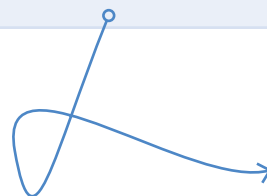
- > When the handset is on the phone and a number is dialed, the corresponding appliance is switched on or off, depending on its current state.
- > When the handset is lifted, the dial can be used to (laboriously) compose a text message in a way similar to mobile phones from some 20 years ago. Replacing the handset on the phone will send the message.
- > The bell of the phone is available to the home automation controller (HAC) as an alarm and can, for example, be routed to the doorbell or function as a kitchen timer or (unpleasant) wake-up alarm.
- > The loudspeaker of the phone is connected to an MP3 player to play prerecorded messages or music. This allows for implementing, for instance, a talking clock, as was common in the previous century.

Communication between the phone and the HAC is based on MQTT, while mDNS is used to automatically establish a connection between the two.



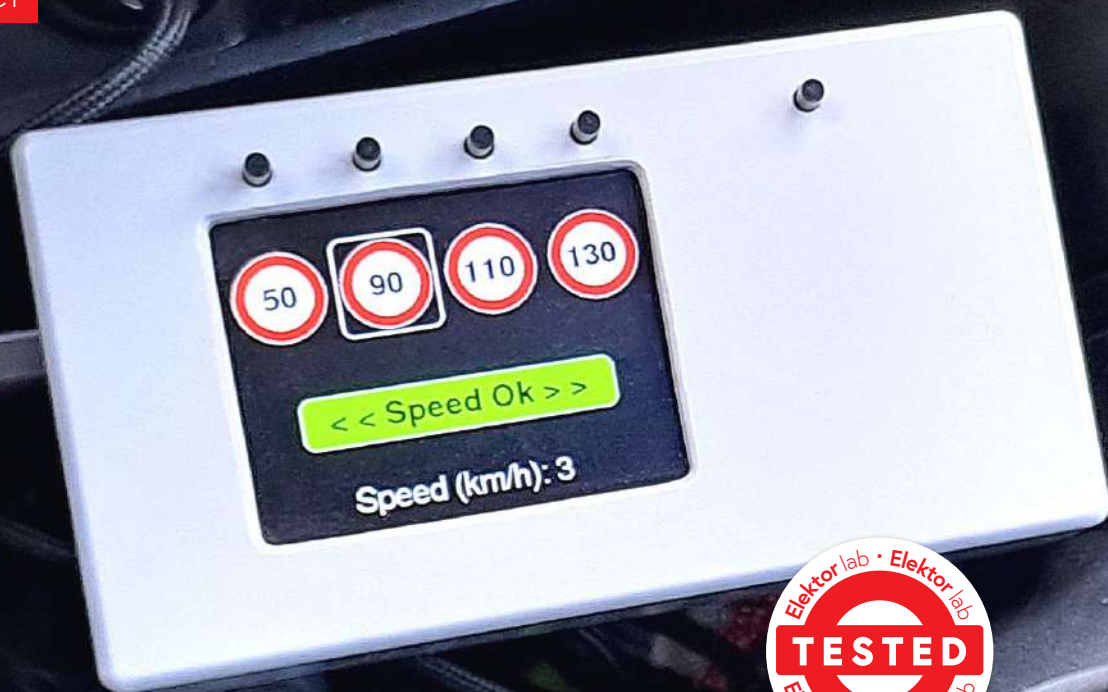
Related Products

- > **WIZnet W5100S-EVB-Pico RP2040-based Evaluation Board (SKU 19971)**
www.elektor.com/19971
- > **WIZnet Ethernet HAT for Raspberry Pi Pico (SKU19970)**
www.elektor.com/19970
- > **C. Valens, Mastering Microcontrollers Helped by Arduino (3rd Edition) (SKU 17967)**
www.elektor.com/17967



WEB LINKS

[1] This project at GitHub:
https://github.com/polyvalens/rotary_dial_remote



GPS-Based Speed Monitor

No More Speeding Tickets

By Olivier Croiset (France)

There are many examples on the internet showing how to use a GPS module in a microcontroller project. But what to do with GPS coordinates, except to plot them on a map? I wanted to make something more useful and came up with this GPS-Based Speed Monitor. This device can be used in any vehicle like a car or a boat. It allows you to drive without having your eye on the speedometer and to avoid unpleasant surprises afterward in the mail.

The device presented here is not a speed limiter, which would require modification to the vehicle, but an alarm that signals when a pre-selected speed is reached (and passed). In order for you not to take your eyes off the road, the buzzer sounds two long beeps if the selected speed is exceeded, while two short beeps indicate that the vehicle has slowed down to a speed below the limit.

Four Presets

The Speed Monitor has four programmable speed limit presets, which should be sufficient for most trips and limitations encountered. Moreover, the screen hardly allows displaying more than four limits. While driving, the driver should be able to choose a preset quickly, without looking at the device at all, if possible. I did not want to use a touch screen; the driver should physically feel the button presses. For ergonomic reasons, the buttons are placed above the four speed limits displayed on the screen.

A Good Excuse to Have a Go at 32 Bits

My first programs using an 8-bit ATmega328 microcontroller associated with a GPS module quickly showed me that its 32 KB of flash memory would not be enough for a more advanced application, especially if the data is to be displayed graphically on a TFT screen.

Therefore, after having exploited the many qualities of the ATmega328, I decided to move on to the next level and try out an STM32 32-bit microcontroller. Its best suited form for tinkerers like me is the BluePill board with its STM32F103C8 device. This little board greatly facilitates the use of this microcontroller. It can be programmed easily through its USB port using the Arduino IDE (provided that the board is programmed with a suitable bootloader beforehand), it is compact, and it is inexpensive (only a few euro).



Flash Memory Sizes

This project is a first attempt to use the STM32F103C8. Note that, officially, the C8 is fitted with 64 KB of flash memory, while the C8 version carries 128 KB, but sometimes the C8 too comes with 128 KB (counterfeit parts?). This will allow, for those who are tempted, to continue learning by adding other peripherals, such as an SD card, a SIM module, data transmission over a radio link, etc.

The GPS Module

The GPS module used in this project is a NEO-6 module from u-blox. Using it is easy, as the TinyGPS+ library [2] does all the work of decoding the NMEA 0183-formatted data stream output by the GPS module. The main parameters that we obtain this way are:

- UTC date and time;
- Longitude and latitude, in decimal degrees;
- Speed;
- Heading;
- Altitude;
- The number of visible satellites;
- The HDOP value.

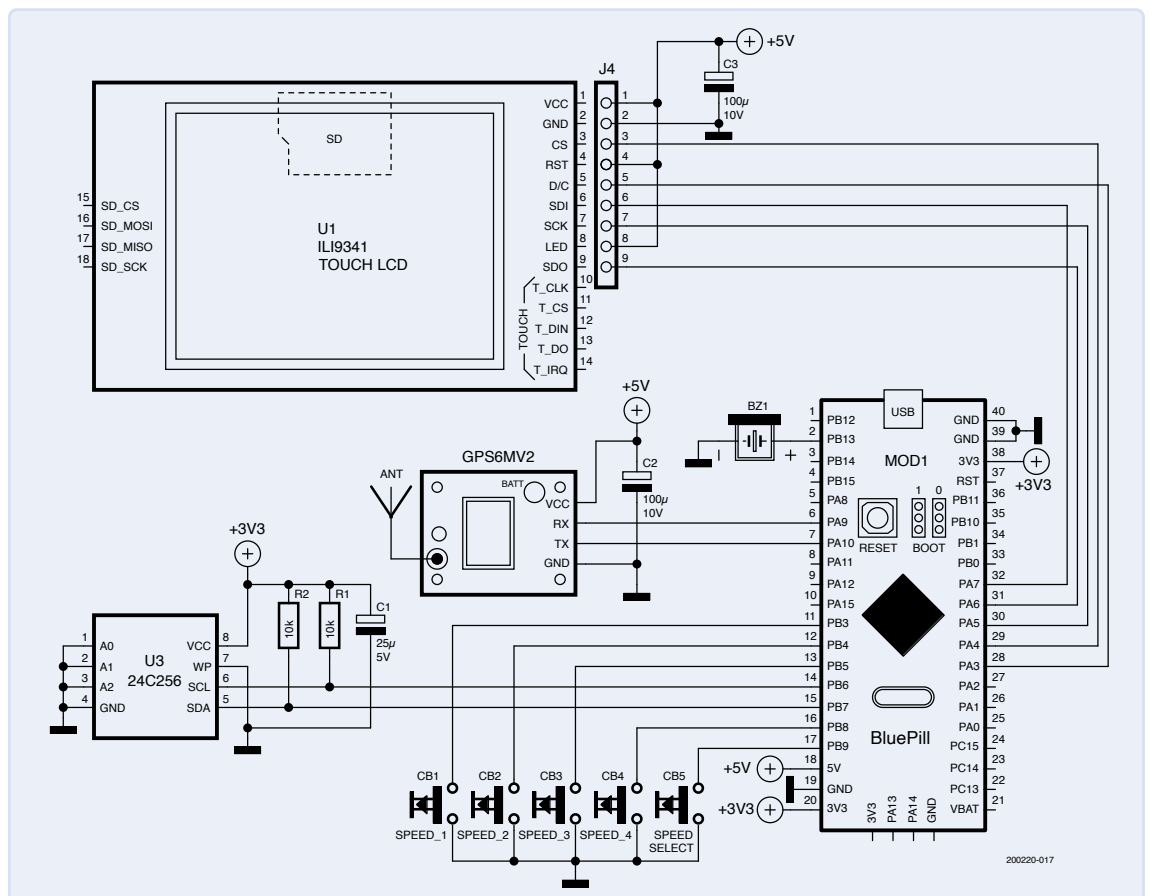
The Horizontal Dilution of Precision, a.k.a. HDOP [2] depends on the position of satellites in range of the GPS receiver. The lower this value is (close to 1), the better the precision of the coordinates. An HDOP value of 10 indicates that the coordinates are not very accurate or even invalid. The Speed Monitor does not use this parameter.

The TFT Display

As display, I used a common 2.2" TFT display module with ILI9341 driver chip and SPI interface. It has a resolution of 320 × 240 pixels, which is sufficient for the data we want to display (four speed limits with a brief message indicating whether the speed limit has been reached or not).

The Speed Monitor has two main display modes during normal use. The current display mode is saved in the EEPROM and is recalled when the system is turned on again. The speed limit presets are also stored in the EEPROM. Only six bytes of this EEPROM are used: four bytes for the speed limits, one byte for the last selected screen, and one byte for the last selected speed limit.

Figure 1: The Speed Monitor combines readily available modules with a few electronic components.



The Circuit Diagram

The schematic of the Speed Monitor is shown in **Figure 1**. The BluePill module in the lower-right corner is the heart of the circuit.

The power supply is connected through the BluePill's USB connector. Newer vehicles are often equipped with a USB connector; otherwise, a 12 V-to-USB adapter is required (or use a powerbank). The BluePill module is powered from 5 V; it has its own 3.3 V regulator. The latter supplies the EEPROM with 3.3 V. The TFT screen and the GPS module are supplied with 5 V. The total consumption of the circuit is about 200 mA.

The display is controlled over a SPI bus, the GPS module is controlled over a serial link, and the EEPROM is connected to the I²C bus (with its two pull-up resistors). For my prototype, I used a 32 KB 24C256 EEPROM, but a 128-byte 24C01 would already be more than big enough.

Twelve GPIO ports remain free. It is thus possible to add more peripherals, features and functions, as long as the application program fits inside the flash memory. The compiled firmware of this project occupies 64,020 bytes, which is almost the entire flash memory.

Building a Speed Monitor

The most important criterion to consider when assembling this project is ergonomics. The driver must be able to find the buttons without ambiguity. The push buttons have a 25 mm lever — the longest I could find.

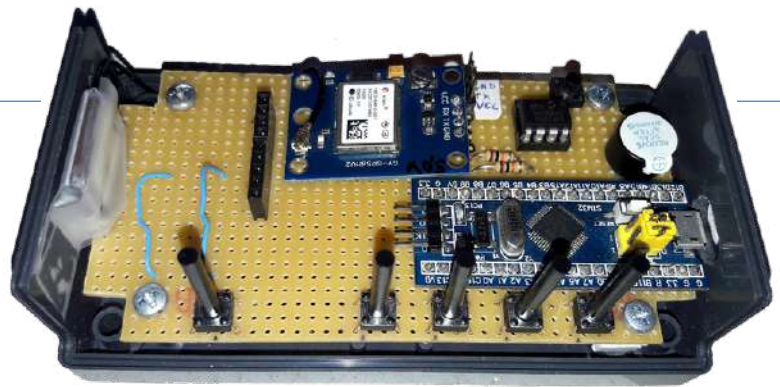
I made this prototype on a prototyping board (**Figure 2**). A printed circuit board (PCB) would allow a more compact realization in a thinner case. The USB connector must remain accessible, as it is used to power the circuit in normal use. It is also needed to update the software.

The GPS antenna should be as far away from the TFT display connector as possible (at least a few centimeters).

Notes on the GPS Module

The GPS module has a tiny data backup battery. If the GPS is not used for a few days in a row, the battery discharges, and the data is lost. When you turn the GPS on again, you may have to wait a few minutes to restore the data (and recharge the battery). The GPS module's LED flashes when it successfully decodes the satellite data. A few moments later, the number of satellites is displayed.

The GPS antenna must “see” the satellites. The reception of the signals must therefore be hindered as little as possible. In urban areas, between two buildings, reception can be difficult. Nevertheless, the speed monitor placed at the passenger's feet in my car works properly.



Prepare the Speed Limits

This should only be done when the vehicle is stationary! At the first power up of the Speed Monitor, the speed limits are all set to 255. This corresponds to FF in hexadecimal, which is what a blank EEPROM contains.

In Screen 3 (**Figure 5**), which shows the data from the satellites, press the third button to activate the speed limit setting. Then, use the first and second buttons to set the desired limit. Press the third button again to validate the speed limit and move on to the next, and so on. Make sure that the limits are in a logical order, as the software does not sort them.

Using the Speed Monitor

The Speed Monitor is easy to use. However, as it is supposed to be used in a moving vehicle, make sure that the device and especially its power cable are not

Figure 2: The prototype was built on perfboard. The enclosure dimensions are 12 x 6.5 x 4 cm.

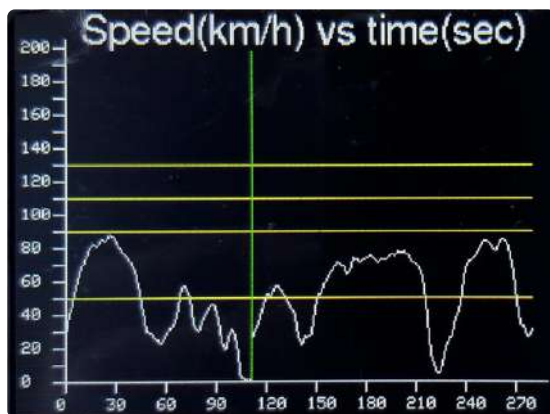


Figure 3: Screen 1 shows the speed limits as round traffic signs together with a brief message.



Figure 4: On Screen 2, the actual speed is presented on a simulated 7-segment display.

Figure 5: The speed history curve is shown on Screen 5. The four yellow horizontal lines show the four programmed limits, and the green vertical line shows the current speed.



in the way while driving. Once it is placed in a suitable position, select the desired display mode. Screen 1 (Figure 3) shows round traffic signs, Screen 2 displays the current speed as 7-segment digits (Figure 4). Then, select the speed limit you want to use. The buzzer will sound if you are going too fast.

Your passenger can view the speed curve over a 4.5-minute (270-second) period by selecting Screen 5 (Figure 5). On this screen, the buzzer doesn't sound. The horizontal lines show the four speed limits. The vertical line indicating 'now' shows the current speed. A scrolling graph might have been prettier, but requires a full refresh for every new data point, and that is slow. Moving the cursor is much easier.

GPS Data Screen

Screen 3 (Figure 6) shows the GPS data and allows you to locate yourself on a map with coordinate markers. The accuracy is about 30 meters, which is quite good for our small and low-cost device. In this display mode, the buzzer does not sound. Screen 4 (Figure 7) is the same as Screen 3, except that it lets you set the speed limits.

Note that the GPS date and time are UTC values, i.e. at the Greenwich meridian. If you want to use them, you should convert these values to your local time zone and the summer/winter time. As the Speed Monitor is not a clock, it doesn't provide a convenient menu for this.

The Software

The program is easy to modify and the source code can be downloaded from [3]. It is an Arduino sketch, and requires the STM32 Boards Package for Arduino (we used v2.3.0) [4]. The lion's share of the code is dedicated to the graphical user interface, and the rest takes care of the simple task of receiving GPS data, extracting the speed and comparing it to a limit.

Figure 6: GPS data is available on Screen 3. This is also the screen that provides access to the speed limit definitions on Screen 4.



Figure 7: Screen 4 lets you set the speed limits. Here, the first limit, 'Sp. 1', is being set (in red, if you look closely).

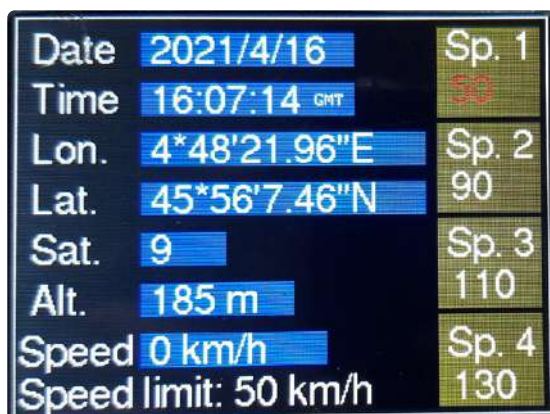


Table 1: The different screens and how to navigate through them.

Screen	Button 1	Button 2	Button 3	Button 4	Button 5
Screen 1 (Round traffic signs)	Speed limit 1	Speed limit 2	Speed limit 3	Speed limit 4	Go to Screen 2
Screen 2 (7-segment display)	Speed limit 1	Speed limit 2	Speed limit 3	Speed limit 4	Go to Screen 3
Screen 3 (GPS data)	Speed limit 1	Speed limit 2	Go to Screen 4	No action	Go to Screen 1
Screen 4 (Define limits)	Decrease limit (-5 km/h)	Increase limit (+5 km/h)	Validate and move to next limit, return to Screen 3 when done	Go to Screen 5	No action
Screen 5 (Speed curve)	No action	No action	No action	Go to Screen 2	No action

As usual in an Arduino sketch, the `setup()` function takes care of initializing all peripherals. When done, it displays a welcome splash screen for five seconds.

The `loop()` function starts by reading the push buttons before checking the GPS for fresh data (in the `dataDecode()` function). Then, depending on which screen is active, the corresponding data is printed. If the current speed is higher than the selected speed limit, an alarm is sounded.

Note that debugging and status information is being sent via the serial port (115200,N,8,1).

To Go Further

Here are a few things that you may want to add or change:

- Use other speed units. The GPS library allows you to use other speed units, such as knots or miles/hour (MPH). This will require changing the screens as well (e.g. replace km/h with knots) and the step size of the limit increments (in the `SpeedSettings()` function). This can be useful for a boat.

If your BluePiL module has 128 KB of flash memory, additional functions can be added, such as:

- Recording to SD card. On the back of the TFT display module is an SD card slot. It can be used to record, for instance, the GPS data for access on a PC.
- Add a touchscreen for new functions. In this case, be sure to take care of the user's safety.
- Setting the local time. ◀

200220-01

Questions or Comments?

If you have technical questions, feel free to e-mail the Elektor editorial team at editor@elektor.com.



Related Products

- **OPEN-SMART GPS – Serial GPS Module for Arduino (SKU 18733)**
www.elektor.com/18733
- **2.2" SPI TFT Display Module ILI9341 (240x320) (SKU 18419)**
www.elektor.com/18419
- **Majid Pakdel, *Advanced Programming with STM32 Microcontrollers* (SKU 19520)**
www.elektor.com/19520



WEB LINKS

- [1] TinyGPS++ library: <http://arduiniiana.org/libraries/tinygpsplus/>
- [2] What exactly is HDOP?: [https://en.wikipedia.org/wiki/Dilution_of_precision_\(navigation\)](https://en.wikipedia.org/wiki/Dilution_of_precision_(navigation))
- [3] Project files at Elektor Labs:
<https://elektormagazine.com/labs/save-money-with-this-speed-monitoring-by-gps>
- [4] STM32 Boards Package for Arduino:
https://github.com/stm32duino/BoardManagerFiles/raw/main/package_stmicroelectronics_index.json

RGB Stroboscope with Arduino

A Colorful Adaptation of a Useful Instrument

By Roel Arits (The Netherlands)

In days gone by, before the advent of modern electronics under the bonnet, stroboscopes were used to adjust the ignition timing of engines. These days this is rarely necessary – in reasonably modern vehicles the ignition timing is regulated by an engine control unit (ECU) aided by countless sensors.

Everything used to be different – not necessarily better, but still different. Not all that long ago, there was still reason enough to get under the bonnet of a vehicle to tinker, make repairs and perform adjustments. Thanks to modern computer-based design methods, it is now possible to fit all the components under the bonnet so compactly and close together that even replacing a simple lamp is a major challenge. And, of course, computers in the form of a large number of interconnected microcontrollers have also found a place under there too.

Let's take a walk down memory lane, however, to the 1970s or 1980s. For proper operation of an internal combustion engine (petrol engine), the spark plugs of all the cylinders have to generate sparks at precisely the right time. This was handled by the distributor (a mechanical contraption, which was perilously sensitive to moisture in some vehicles), that had to be precisely synchronised with the engine. That's where the stroboscope – or timing light as it was also called back then – came into play.

An old-fashioned distributor had the same number of contacts as the number of cylinders in the engine and, each time a contact opened, the electrical energy stored in the ignition coil produced a spark at the tip of the spark plug. In turn, this ignited the fuel-air mixture in the cylinder. That's what made the engine run.

The stroboscope used to adjust the timing had a xenon or neon flash

lamp and a trigger cable. The trigger cable (with an inductive sensor) was attached to the reference spark plug cable so that each time the reference spark plug fired, the flash tube was triggered and produced a very short flash of light. There was a white mark (a narrow stripe) on the crankshaft pulley with a matching fixed mark on a sort of angle bracket above the pulley. These two marks were supposed to be exactly aligned to each other when the reference spark plug fired, and the stroboscopic effect of the timing light made it look like this moving mark was standing still. If the moving mark on the crankshaft pulley was not aligned to the fixed mark, but instead offset to the left or to the right, this indicated that the distributor required adjustment.

Triggering? Who Needs It?

In the application described above, the stroboscope is synchronised by a trigger signal with the rotary motion being measured. However, a trigger signal is not always necessary.

If you want to measure the engine speed (RPM) using a non-synchronised stroboscope, you have to adjust the frequency of the light flashes so that the mark on the pulley appears to be practically stationary and can be seen in only one place. If several nearly stationary marks can be seen at the same time, this means that the flash rate is a multiple of the RPM, or the RPM is a multiple of the flash rate.

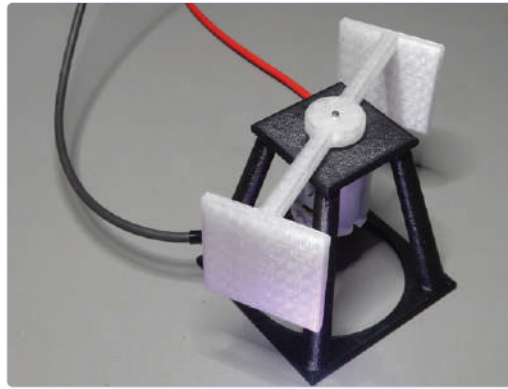


Figure 1: The frame with the propeller that serves as a reflector.

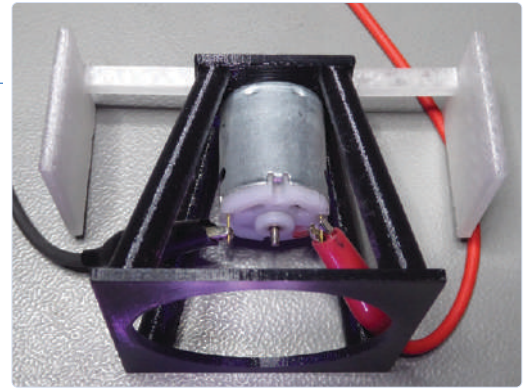


Figure 2: Close-up of the motor.

If the pulley is turning clockwise and the mark is drifting in the clockwise direction, the flash rate is a bit too low. Each flash comes too late, so the mark appears to move in the direction of rotation. If the pulley is turning clockwise and the mark is drifting in the anticlockwise direction, the flash rate is a bit too high, so each flash must be a bit too early. In this way you can measure the RPM by adjusting the flash rate of the stroboscope so that only one stationary mark is clearly visible. Each flash must be short enough to produce a distinct reflection (in other words, a distinctly visible mark). If the flash duration is too long compared to the flash rate, the mark will look 'smeared'.

Let's Have Some Fun with This

What happens if we generate several flashes at the same rate but with different colors and a phase shift between the individual flashes? And, what will we see if we also use a rotating white object and vary the frequency, phase shift, and duration of the different colored flashes? This is not difficult to achieve with the aid of a microcontroller, so there's nothing stopping us from trying this. We can use an RGB LED for the different colored flashes or, better yet, three separate LEDs to produce brighter flashes. To control the flash rate, phase shift, and flash duration, we can use an Arduino Pro Mini as it has enough I/O to work with and is more than fast enough for what we have in mind. The LEDs, as you might expect, can be driven by pulse-width modulated (PWM) signals.

We could use the three dedicated PWM modules of the Arduino Pro Mini, but they operate from three different timers. That makes it more difficult to synchronize them and to program the phase shifts. What's more, we need relatively low-frequency PWM signals, so a resolution of 16 bits is far too much. Software-generated PWM signals (softPWM) are perfectly satisfactory for our purposes. With softPWM, the signals are output on normal digital I/O pins. These are set and reset using a counter that generates an interrupt each time it reaches a specific state. This allows us to configure an interval that is short enough to adjust the pulse width or phase shift of the PWM signals with sufficient precision.

Putting It Into Practice

To implement the concept described above, the author used a 12 V DC motor (Velleman part number MOT3N) and designed a small frame for it. The motor turns a sort of propeller with two vertical blades that act as reflectors. Both parts were made with a 3D printer and the relevant files are included in the download for this project. **Figure 1** and **Figure 2** show what they look like.

Note that the no-load speed of the motor used by the author is around

11,500 rpm, which is much too fast for the 3D-printed propeller. The first time the author tested this setup, parts were literally flying past his ears. This is naturally a bit risky, so it is a good idea to wear safety glasses or goggles. You have been warned!

For this reason, the author operated the motor from a DC voltage of approximately 3 V. This voltage can be adjusted to synchronize the motor speed to the flash rate. With a 3 V supply voltage, the motor speed is approximately 2100 rpm, equivalent to a frequency of 35 Hz. The propeller has a symmetrical structure with two reflecting blade surfaces, which makes balancing a little easier. Because of this, the flash rate has to be twice as fast (70 Hz) corresponding to a period of approximately 14 ms.

But first, let's look at the circuitry. The schematic diagram shown in **Figure 3** is a model of simplicity. The three RGB LEDs are switched on and off by modest driver transistors under the control of the Arduino Pro Mini. The supply voltage for all of this is 5 V DC, with a current consumption of approximately 500 mA.

No PCB has been designed for this, but you can build everything on a breadboard in next to no time. The author's setup is shown in **Figure 4**. The stroboscope timing is controlled using a timer interrupt that is triggered every 100 μ s. This 0.1 ms interval is therefore the resolution for adjusting the flash rate and proved to provide adequate control capability for shifting the 'stationary' image of the propeller.

```
...
// Setup 16bit timer1 in normal operation with
// interrupt at 100us
// 16MHz/1 = 6.25ns. So to get 100us we need to let
// the timer count 1600 ticks.
TCCR1A = 0;
TCCR1B = _BV(CS10); //prescaler divide by 1
TCNT1 = 0xFFFF - 1600; // overflow is at 65535 =
// 0xFFFF
TIMSK1 = _BV(TOIE1); // overflow interrupt
TCNT1 = 0;
sei();
}
ISR (TIMER1_OVF_vect)
{
TCNT1 = 0xFFFF - 1600; //100us interrupt
...

```

The flash period is hard-coded in the software and set to 144, which is the number of timer interrupts between two successive flashes. This means the period is $144 \times 100 \mu\text{s} = 14.4 \text{ ms}$. The flash duration for each

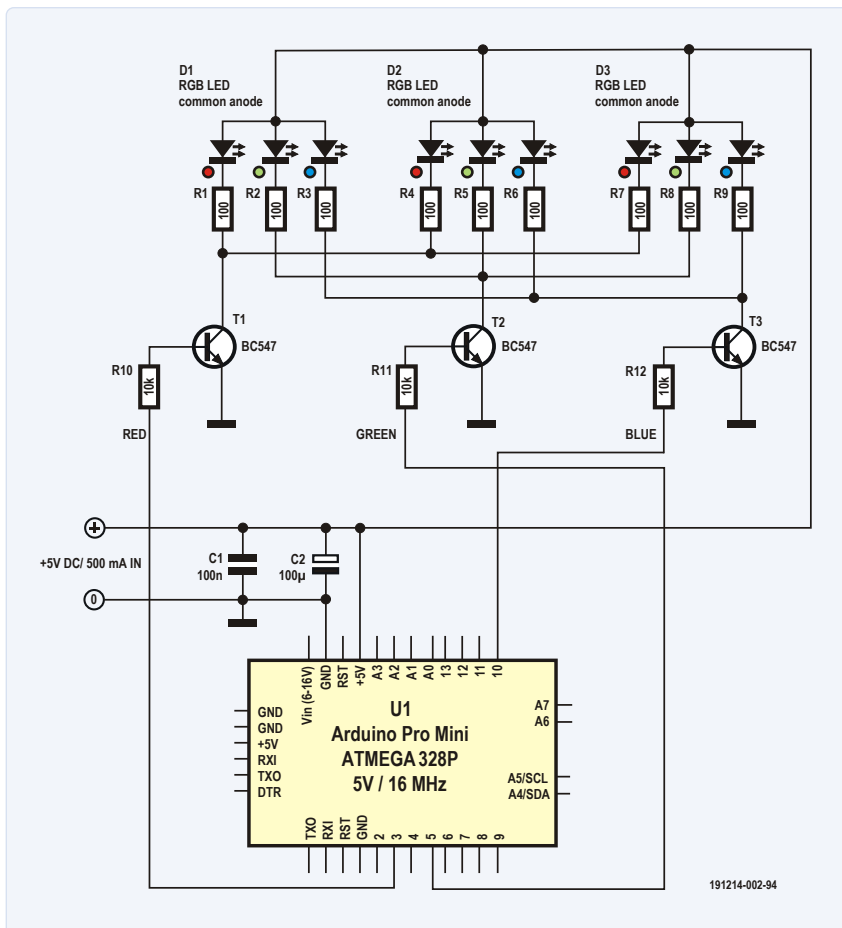


Figure 3: The schematic is fairly simple.

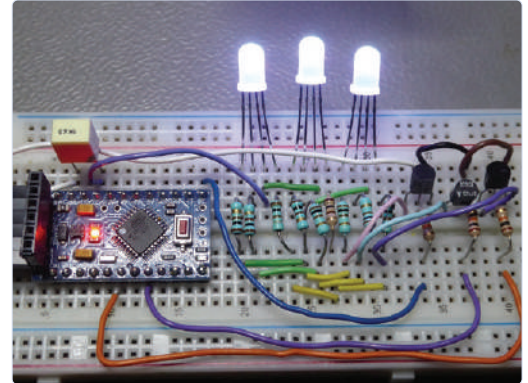


Figure 4: The construction is not critical and a breadboard is fine for initial experiments.

LED is set to a fixed value of 8, corresponding to $8 \times 100 \mu\text{s} = 800 \mu\text{s}$. The duty cycle (mark/space ratio) is therefore $800 \mu\text{s} / 14.4 \text{ ms} = 5.5\%$.

```
#define STROBE_PERIOD 144
```

```
TSoftPwm Pwm[] = {TSoftPwm(ID_RED, 0, 8,
    STROBE_PERIOD),
    TSoftPwm(ID_GREEN, 0, 8, STROBE_PERIOD),
    TSoftPwm(ID_BLUE, 0, 8, STROBE_PERIOD)};
```

This duty cycle gives a nice sharp image once the motor speed is suitably adjusted.

The circuit plus software (which can be downloaded for free from the project page for this article [4]) is not a finished project. Instead, the author intends it to be a proof of concept and has therefore not provided a nice user interface or any other fancy features. The available software demonstrates what can be done by playing with the parameters of flash rate, flash duration, and phase difference. The software is extensively and clearly commented, so there's no need to describe it in detail here.

Of course, a non-synchronized stroboscope leaves room for improvement. In principle, it shouldn't be that difficult to mount a small light beam interruption sensor next to the propeller and use the sensor signal to trigger the stroboscope. However, we'll leave that as a project for our interested readers.

The author has posted two videos on YouTube where you can admire the operation of the stroboscope [1] and view the interaction between the LED control signals on an oscilloscope screen [2]. The project is also available on the Elektor Labs site [3].

191214-01



Related Products

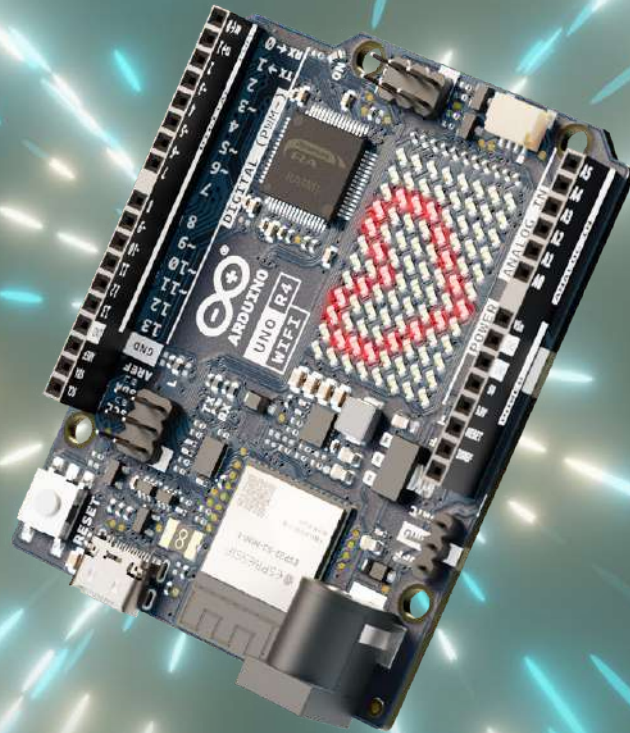
- > **SparkFun Arduino Pro Mini 328 (5 V, 16 MHz)**
<https://elektor.com/20091>
- > **Günter Spanner, Home Automation Projects with Arduino**
<https://elektor.com/18128>

WEB LINKS

- [1] Demo video 1: <https://youtu.be/WHv6DCWM82k>
- [2] Demo video 2: <https://youtu.be/3tYqUin4-vQ>
- [3] Project page on Elektor Labs: <https://elektormagazine.com/labs/arduino-rgb-color-stroboscope>
- [4] Project page for this article: <https://elektormagazine.com/191214-01>



UNO R4



The new dimension of making

Take a quantum leap with the new UNO R4, the most advanced and powerful UNO ever!



Faster and larger memory - 48 MHz, 256 kB Flash, 32 kB RAM - perfect to perform more precise calculations and handle more complex projects.



New built-in peripherals - The RA4M1 (Arm® Cortex®-M4 Core with FPU) includes a DAC, CAN-BUS, HID support and OpAMP embedded allowing for more advanced projects.



Expand your projects with UNO shields (5 V compatible) and the huge catalog of *Qwiic nodes.



Connectivity *ESP32-S3 coprocessor - Wi-Fi® and Bluetooth® Low Energy connectivity, leaving the RA4M1 microcontroller free.



Robust by design - with a larger voltage range, 6-24 V, now you can use the UNO R4 in combination with motors, LED strips or other actuators from a single power supply.

UNO R4 WiFi



UNO R4 Minima



*Note - these features are only available on the UNO R4 WiFi model.



Wireless Emergency Push Button



Enhanced Safety with LoRa

By Somnath Bera (India)

The Wireless Emergency Push Button described in this article is an example of how a problem in a complicated industrial environment can be solved with a few low-cost electronic modules and some Arduino programming. The result is a simple wireless system that can be useful in many other situations.



The original need for developing the wireless Emergency Push Button (EPB) presented here is rather specific and has to do with taking samples from trains delivering coal to an electric power plant. A coal sample must be taken from the top of a coal wagon to determine its calorific value before it may be used. This is an important parameter for a power plant.

Before taking a sample, the sampler presses the existing EPB to signal his presence to the driver of the train. This should immobilize the train. Unfortunately, depending on the length of the train, the sampling position, and the curvature of the track, the sampler cannot always see his signal in front of the train. Occasionally while collecting samples, it has been observed that the train rolled inadvertently, causing accidents, and hurting the person. The EPB system needed improvement.

Things to Be Aware Of

The coal trains can be very long, up to 500 meters, and there can be up to four of them next to each other being sampled and unloaded at the same time. Add to that curved railroad tracks, and you will understand that it is complicated to see what is going on. Furthermore, the environment is noisy due to unloading trains. Audible feedback is thus difficult, too.

The existing EPB system is a wired system. However, in such an environment, having cables run over long distances is risky, as they may break without anyone noticing. Replacing them or running more cables to provide visible feedback is complicated and expensive. Increasing the height of the signal mast so that it can be seen from a distance is not a practical solution either, as it requires good eyesight to identify for which track it would be.

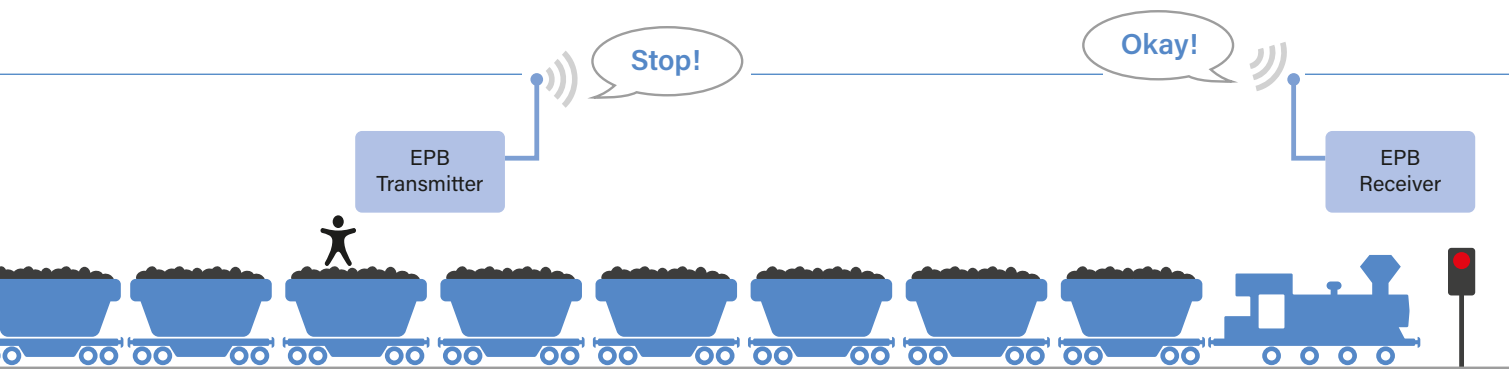


Figure 1: Overview of the Wireless Emergency Push Button (EPB) System.

Going Wireless with LoRa

The solution we came up with is the wireless EPB (**Figure 1**). It consists of two units: an EPB Sender and an EPB Receiver. Actually, the units are almost identical and capable of sending and receiving, but we will refer to them in this way. If you prefer, you can think of it as a master-slave or client-server system.

Before taking a coal sample, the sampler presses a button on the EPB Sender to signal his request to sample. When the EPB Receiver in the control cabin receives this request, it activates a relay to press the button on the existing EPB system, and it sends an acknowledgement back to the EPB Sender unit. Now the sampler knows that his signal has hit home, and he can safely take a sample. When done, the sampler presses a second button on the EPB Sender to release the system. This time, the EPB Receiver releases the existing EPB by activating a second relay and again confirms this to the EPB Sender. The system is ready for a new sample.

In case the system is not available, or the signaling EPB is kept bypassed for operational purposes, the feedback signal will never return to the sender unit, alerting the sender and avoiding miscommunication.

EPB Sender and Receiver Circuits

The EPB Sender is an exact replica of the existing EPB unit, but with a small antenna and a small OLED screen added to it. It is powered by a single-cell LiPo battery. It has two push-to-activate buttons. Inside (**Figure 2**) is an ESP32 module that drives the OLED display and a LoRa transceiver module. Note that the ESP32 module was only used for convenience, not for its wireless capabilities. Any other cheap, low-power microcontroller module with the right interfaces (I²C, UART, 2x GPIO) can be used instead.

The EPB Receiver is similar to the sender unit, except that it has two relays instead of the push buttons, and it doesn't have a display (**Figure 3**). Note that only one of the relays should be active at any time. This is taken care of by the software.

Security

To have improved security such that the system never gets disturbed by some spurious signal, we selected low-power LoRa transceiver modules from Ebyte. Besides using interference-resistant spread-spectrum technology, they provide three parameters: channel frequency, air rates and a 4-byte ID. Communication can only take place when these three parameters are the same on both ends of the link.

Figure 2: The EPB Sender consists of an OLED display and two push buttons (besides the LoRa transceiver).

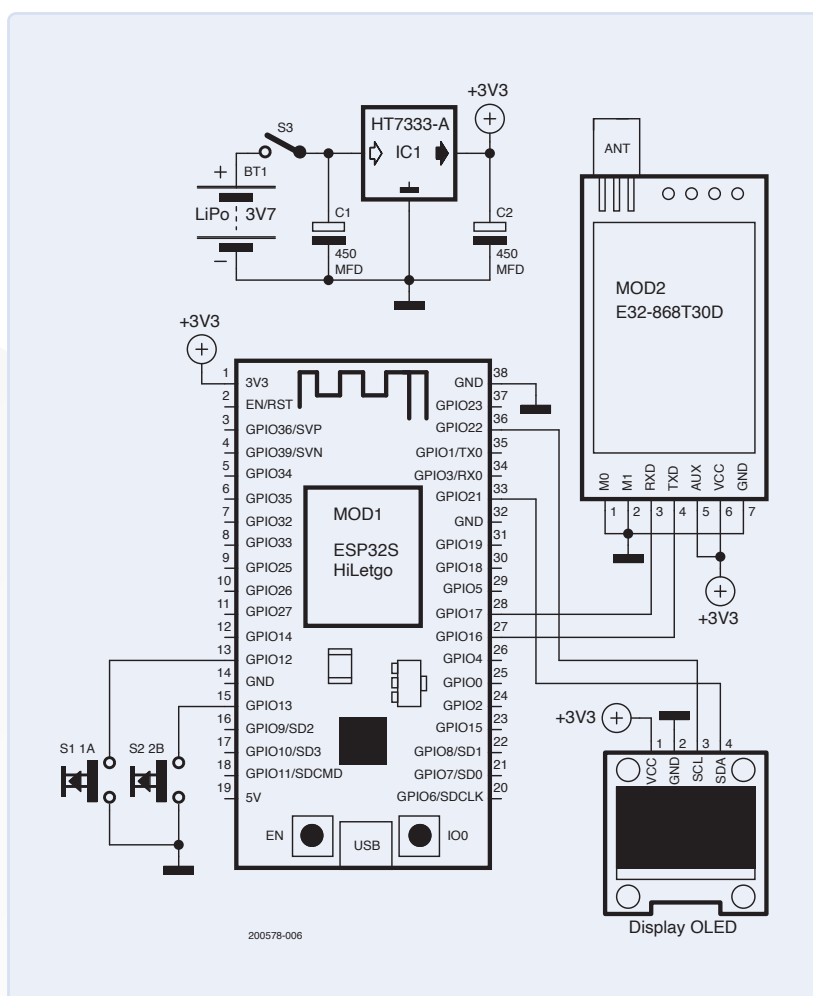


Figure 3:
The two relays
of the EPB
Receiver press
the buttons on
the existing
EPB system.

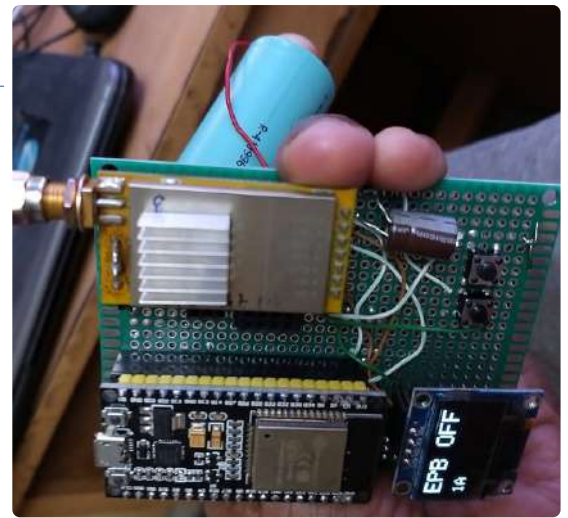
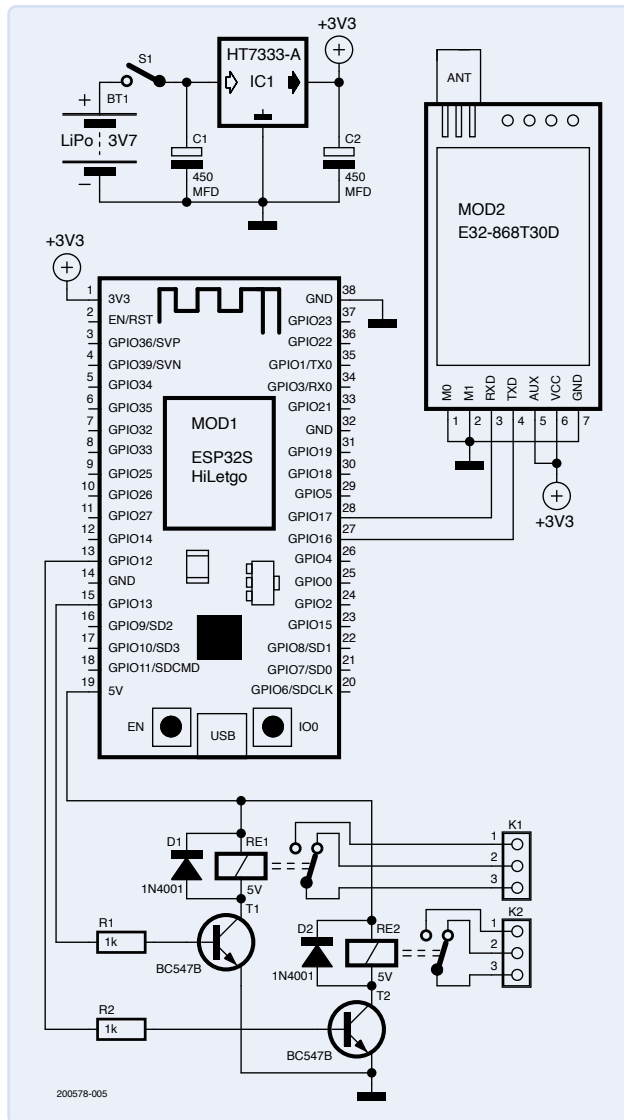


Figure 4: Prototype of the EPB Sender built on perfboard.

In the `loop()` function, both the sender and receiver first check if a message has been received from the other unit. If so, they update their state accordingly. The receiver sends an acknowledgement message back to the sender and starts waiting for a new message.

The sender continues by checking the state of its two push buttons. Depending on which one is pressed, it sends a short ASCII string over the serial port to the LoRa transceiver. Only one key press is accepted at any time.

The software can be downloaded from this project's page at Elektor Labs [1]. Feel free to modify it as you see fit. The strings exchanged between the two units were chosen more or less arbitrarily and may be replaced by anything you like. ◀

200578-01

The module features a serial interface and delivers up to 500 mW (21...30 dbm) at 868 MHz (Figure 4). The signal can travel 500 meters very comfortably along the railway track with a small handheld antenna and with the receiver unit antenna placed on the rooftop of the cabin / control room so that it remains in sight everywhere.

Software

Like the hardware of the two units, the software for the units is similar as well, and consists of two short Arduino sketches.

Both units configure the serial port used for communicating with the LoRa transceiver as 9600N81 in the `setup()` function. The EPB Sender configures two GPIO pins as push button inputs, whereas the EPB Receiver configures them as outputs to drive relays. The EPB Sender also prepares the OLED display.

Questions or Comments?

Do you have technical questions or comments about his article? Contact Elektor at editor@elektor.com.



Related Products

- ▶ **ESP32 DevKitC (SKU 18701)**
<https://elektor.com/18701>
- ▶ **0.96" 128x64 I2C OLED display (SKU 18747)**
<https://elektor.com/18747>
- ▶ **Claus Kühnel, Develop and Operate Your LoRaWAN IoT Nodes (SKU 20147)**
<https://elektor.com/20147>

WEB LINK

- [1] Project files at Elektor Labs:
<https://elektormagazine.com/labs/wagon-top-coal-sampling-remote-epb-module>



Starting Out in Electronics...

Follow the Emitter

By Eric Bogers (Elektor)

In the last episode we briefly focused on transistors as amplifiers, as opposed to switches. Now we turn our attention to what is known as the emitter follower, which is about the simplest imaginable transistor circuit.

You need to reduce the load on the master potentiometer as much as possible, and you must decouple the individual channel potentiometers from each other. This can be done with emitter followers, as shown in the lower part of Figure 1. This works as follows:

When a transistor is conducting, the voltage on its emitter is approximately 0.7 V lower than the voltage on its base, so the control range is from 0 V to 11.3 V. That is a bit more than the nominal 10 V of this particular example, but it has the advantage that when the controls

The Common-Collector Circuit

The first amplifier circuit we will look at is formally known as the common-collector circuit. It has this name because the transistor collector is the common reference point for the input and output signals. This designation is not especially illuminating, so this type of circuit is better known by the name *emitter follower* — a good description of what the circuit actually does: The voltage on the transistor emitter follows the voltage on the transistor base. Read on to learn what this is good for.

Suppose you want to build a mixer board for a lighting installation — nothing too professional, just something simple for home use and operating at a low voltage. For example, you could have a number of lamps (channels) with their brightness individually adjustable (from 0 V to 10 V in this example), along with an additional master control so that all the lamps can be dimmed at the same time. As a beginner, you might come up with the idea of using a master potentiometer and connecting channel potentiometers to its wiper for each of the lamps. This approach is shown in the upper part of **Figure 1**.

However, this is not the right way to go. The channel potentiometers collectively form a heavy load and will seriously distort the control curve of the master potentiometer, and, what's worse, all these potentiometers will influence each other, making effective control impossible.

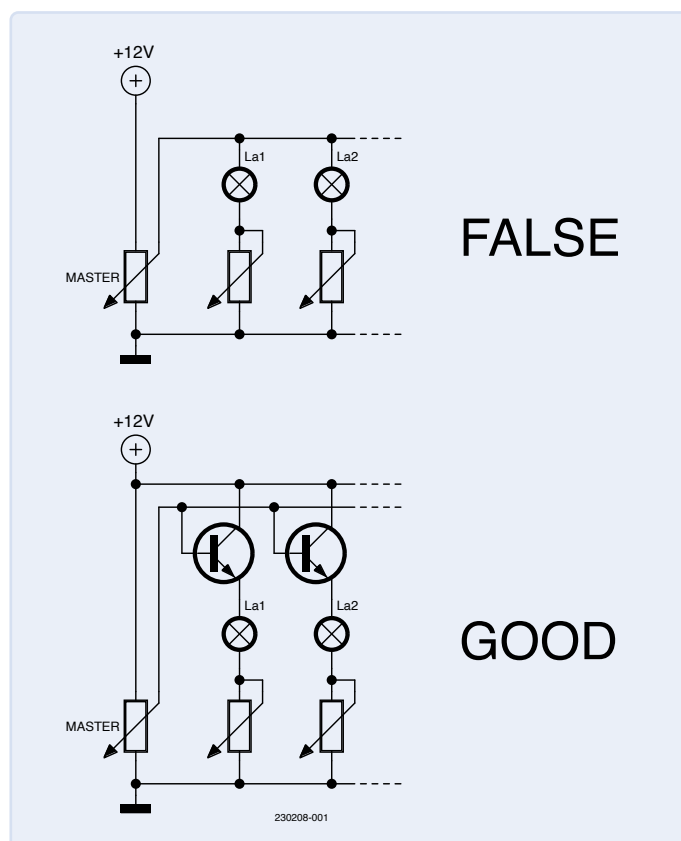


Figure 1: A highly simplified light mixer board: the wrong way at the top, the right way at the bottom.

are set to max, the lamps are actually lit at their maximum brightness. When the voltage on the base is less than 0.7 V, the transistor is cut off and the output voltage is 0 V.

Between these two limits, the voltage on the emitter nicely follows the voltage on the base, but with an *offset* equal to the value of the base-emitter voltage. The main advantage of using an emitter follower in this situation is that the base current places only a very small load on the wiper of the master potentiometer, since the base current is much smaller than the emitter current. The ratio is equal to the DC current gain (h_{FE}) of the transistor. In practice, this means that the load on the master potentiometer is a factor h_{FE} less than the current set by the channel potentiometer.

As you can see, the common-collector (emitter follower) circuit does not provide any voltage gain, but instead provides significant current gain along with a high input impedance and a low output impedance. An emitter follower has an extremely stable response to load changes. You would normally expect the output voltage to drop when the load resistance decreases, but when this happens the base-emitter voltage U_{BE} increases, so the base current I_B also increases. This causes the collector current I_C to increase, and with it the emitter current I_E , offsetting the decrease in the output voltage.

The following formulas apply to the input (top) and output (bottom) impedance of an emitter follower:

$$Z_{in} = (h_{FE} + 1) \cdot Z_{load}$$

$$Z_{out} = \frac{Z_{source}}{h_{FE} + 1}$$

A Stabilized Power Supply

In an earlier episode, we did some calculations to show that using a Zener diode and a series resistor to stabilize a voltage has some significant disadvantages. This sort of stabilization circuit works considerably better if you include an emitter follower, as shown in **Figure 2**.

Let's do the calculations on this circuit using the values from the previous example (a Zener voltage of 48 V and a maximum load current of 14.1 mA, suitable for use as a phantom power supply for a microphone). The base current of the transistor is then given by the following formula:

$$I_B \approx \frac{I_E}{\beta} = \frac{14.1 \text{ mA}}{100} = 0.14 \text{ mA}$$

The DC current gain applies to the collector current. Strictly speaking, the emitter current is equal to I_C plus I_B — but in small signal operation with a current gain greater than 100, we can consider

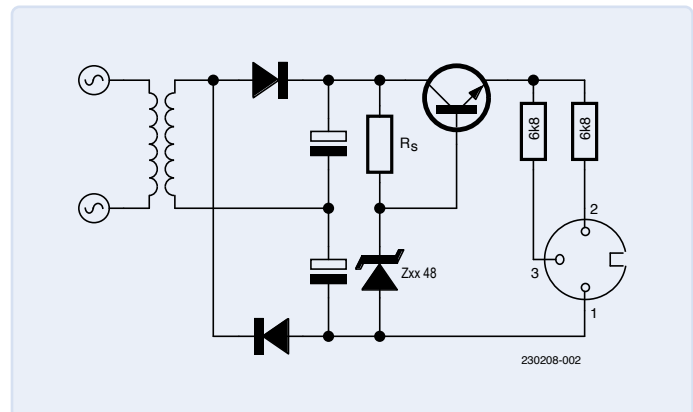


Figure 2: A stabilized power supply (phantom supply for a microphone).

I_C to be nearly the same as I_E . We don't know the exact value of the transistor's current gain, but assuming a gain factor of 100 for small-signal transistors is certainly on the safe side.

During one cycle of the input AC voltage, the voltage on the electrolytic filter capacitors settles to approximately 60 V, which means the voltage over the Zener diode series resistor is 12 V. Then we have:

$$R_{series} = \frac{U}{I_B} = \frac{12 \text{ V}}{0.14 \text{ mA}} = 85 \text{ k}\Omega$$

Here, we take the standard value 82 k Ω for the series resistor.

The maximum dissipation in the Zener diode occurs when no load is connected, so the base current of the transistor is zero. Under these conditions:

$$P_{zener} = U \cdot I = 48 \text{ V} \cdot \frac{63 \text{ V} - 48 \text{ V}}{82 \text{ k}\Omega} = 8.78 \text{ mW}$$

Of course, power is also dissipated in the transistor:

$$P_{transistor} = U \cdot I = (63 \text{ V} - 48 \text{ V}) \cdot 14.1 \text{ mA} = 211.5 \text{ mW}$$

The sum of the dissipation in the Zener diode and the dissipation in the transistor is already less than with the Zener diode alone, since the current without the transistor was a bit more than 900 μ A. The circuit with the transistor also has another advantage: without the transistor, the maximum dissipation occurs when no load is connected and the minimum dissipation occurs when the load is 0 Ω , which, of course, never happens in practice. In the circuit with the transistor, by contrast, the maximum power dissipation occurs with a load resistance of 0 Ω , while the dissipation is less under normal operating conditions.

Using an Emitter Follower As an AC Amplifier

Up to now, we have only used the emitter follower as a DC amplifier. Now, it's time to look at amplifying AC signals (see **Figure 3**).

When an AC voltage is applied to capacitor C1, it alters the voltage on the base (with respect to ground), causing the emitter current to vary such that the voltage at the emitter is always approximately 0.7 V lower than the voltage at the base. Here, as well, the circuit does not provide voltage gain but does provide current gain.

For the greatest possible amplitude range with this circuit, the quiescent emitter voltage needs to be approximately half the supply voltage, which, in this example, means 6 V. For the sake of clarity, the quiescent voltage and quiescent current are quantities that are measured when no signal voltage (AC) is applied.

If we assume a value of 1 mA for the quiescent current, the corresponding value of the emitter resistor is:

$$R_E = \frac{U_R}{I_R} = \frac{6V}{1mA} = 6k\Omega$$

This value (6 kΩ) is in between two E12 values, so here we choose 5.6 kΩ. The base voltage of the transistor is 0.7 V higher than the emitter voltage, so it is 6.7 V with respect to ground. This voltage is set by a voltage divider. For R2, we arbitrarily select a value of 100 kΩ, so we can now calculate the value of R1. This calculation is left as an exercise for the reader; we arrived at a value of 79.1 kΩ and chose a 82 kΩ resistor. As a result, the base and emitter voltages are approximately 0.1 V lower than originally intended, but that is certainly not a problem.

Now, we have to dimension the two capacitors. We want the circuit to have a frequency response that is as linear as possible over the frequency range of 20 Hz to 20 kHz, so we set the lower corner frequency in both cases to 10 Hz. Capacitor C1, in combination with resistors R1 and R2 in parallel, and the input impedance of the transistor, forms a high-pass filter, and, as you now know, the transistor's input impedance depends on the load resistance. If we assume that the load resistance will not be less than R_E, the parallel combination of the load resistance and R_E amounts to 3 kΩ. With an assumed current gain of 100, the transistor's input impedance (R_{in}) is approximately 300 kΩ. The parallel combination of R1, R2, and R_{in} thus amounts to 39.2 kΩ (check this yourself!).

We use the following formula to dimension C1:

$$C_1 = \frac{1}{2 \cdot \pi \cdot f \cdot Z} = \frac{1}{2 \cdot 3.14 \cdot 10Hz \cdot 39.2k\Omega} = 0.4 \mu F$$

With a value of 0.47 μF, we're on the safe side, in any case.

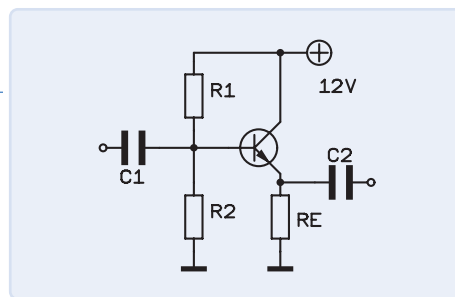


Figure 3: An AC amplifier.

For C2, we do the calculation with an assumed minimum load impedance of 6 kΩ:

$$C_2 = \frac{1}{2 \cdot \pi \cdot f \cdot Z} = \frac{1}{2 \cdot 3.14 \cdot 10Hz \cdot 6k\Omega} = 2.6 \mu F$$

In this case, we should choose a value of 3.3 μF (or 4.7 μF if it is easier to get).

A remark for the fusspots among our readers: In the calculation for R1, we did not consider that the transistor's DC impedance (which is equal to the product of the current gain and R_E) is actually in parallel with R2.

If we want to properly include this in the calculation, we need to know the transistor's current gain reasonably accurately. If we assumed a value of 100 just to be on the safe side, the operating point would be shifted quite a lot; a value of 300 to 500 is more realistic for small-signal transistors. However, in that case, we can ignore the influence of current gain on the calculation for R1.

That's it for this installment. Next time will be really exciting because we'll be exploring voltage amplification with common-emitter circuits — something entirely different to emitter followers. ◀

Translated by Kenneth Cox — 230208-01

Editor's note: The 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?

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



Related Products

► **B. Kainka, Basic Electronics for Beginners (Elektor, 2020) (SKU 19212)**
www.elektor.com/19212

► **B. Kainka, Basic Electronics for Beginners (Elektor, 2020) (E-Book, SKU 19213)**
www.elektor.com/19213

Arbitrary, Independent Hysteresis Levels for Comparators

with Simulations, Spreadsheets and Algebra

By Stephen Bernhoeft (United Kingdom)

Often, one needs hysteresis levels that are both accurate and independent of one another. For the usual comparator configurations, this is entirely possible. Here's how.

Except in the case where switching thresholds are equidistant from the supply rails, the selection of resistor values to accurately satisfy hysteresis levels for comparator circuits is not commonly treated in the available literature. In what follows, some basic algebra is used to derive formulae for both “-ve input” and “+ve input” circuits that are in common use. Related spreadsheets, derivations, and *LTSpice* schematics are available for download [1].

The results presented apply to:

- > Rail-to-rail output devices
- > Open-collector devices where $R_{PULLUP} \ll R_{FEEDBACK}$

To implement an open-collector design, first choose R_{PULLUP} and then make $R_{FEEDBACK}$ (labeled ‘F’ in the rest of this article) at least $10 \times R_{PULLUP}$. The other resistors are calculated based on F.

Parameters

To design the circuit, choose the required trip points and note the output levels. Values are ratiometric and can be scaled, but for open-collector parts, the pull-up value needs consideration.

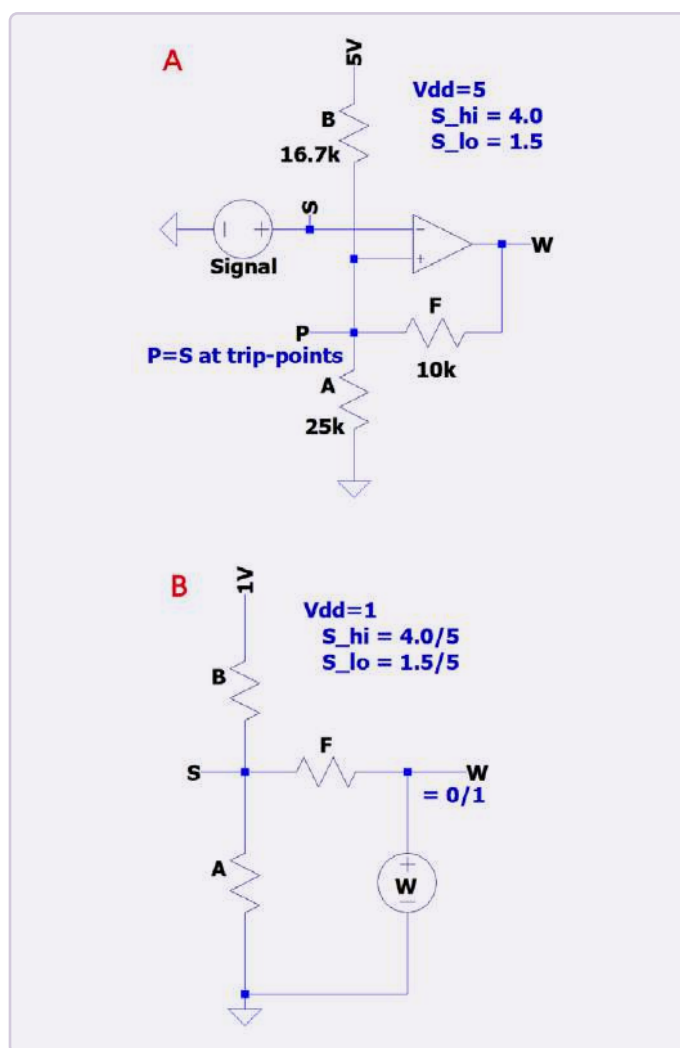


Figure 1: At the top is the real circuit (A), while below it is a simplified version for analysis (B).

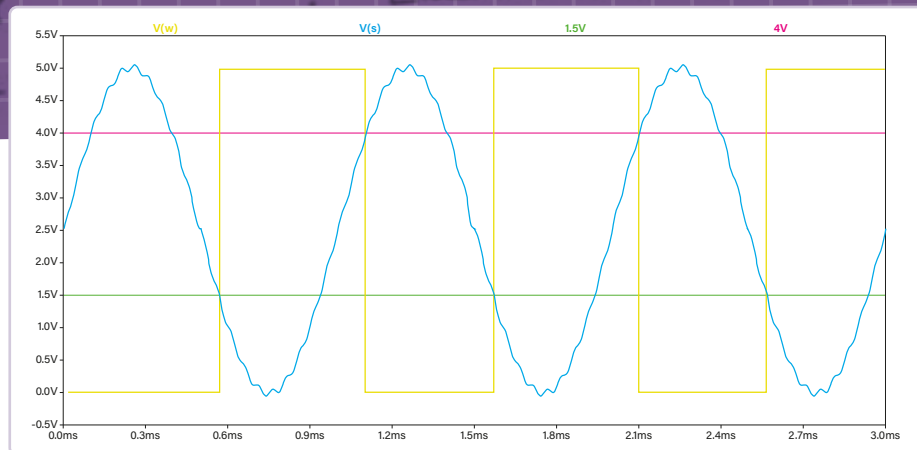


Figure 2: The simulated signals of circuit (A) from Figure 1.

Negative (Inverting) Input Circuit

For the first case, refer to **Figure 1**. The advantages of this configuration are that there is no need for a separate reference and it has a high input impedance and a low component count. The inconvenience may be the inverting action, which might be undesirable, especially when trying to reduce the component count.

In the abstracted circuit in Figure 1B, we see that F is effectively in parallel with B (if $W = 1$) or A (when $W = 0$). The supply is 1 V to simplify the math. Clearly, the 1 V supply can be scaled. For example, a supply of 5 V with a trip-point of 4 V is equivalent to a supply of 1 V with a trip-point of $4/5$ V, or 0.8 V.

W changes state when $P = S$, so we can assume that $P = S$ at either trip-point. In Figure 1A, node P moves according to the comparator output. We can rename P to S in the schematic of Figure 1B because it has the same value as S .

Now, if we define:

- S_{LO} = Lower trip point
- S_{HI} = Upper trip point

And we let:

$$k_{BF} = \frac{S_{HI} - S_{LO}}{S_{LO}} = \frac{B}{F}; \quad k_{AB} = \frac{S_{LO}}{1 - S_{HI}} = \frac{A}{B}$$

Then:

Case 1: Fix A

$$B = \frac{A}{k_{AB}}; \quad F = \frac{B}{k_{BF}}$$

Case 2: Fix B

$$A = B \cdot k_{AB}; \quad F = \frac{B}{k_{BF}}$$

Case 3: Fix F

$$B = F \cdot k_{BF}; \quad A = B \cdot k_{AB}$$

We have now established the ratios $A : B$ and $B : F$ in terms of the trip points. Choosing one resistor (A , B , or F) allows the other two values to be computed. **Figure 2** shows the simulated signals of circuit (A).

Design Considerations

We need to choose trip-points based on several considerations:

- For timing RC circuits: Avoid the extremes of the slow 'tail' of the RC charge curve, because tiny changes in voltage equate to large changes in time.
- Trip levels should be essentially independent of comparator offset voltages. Therefore, the lowest trip-point must be much larger than the (absolute) value of the offset voltage.

Using a spreadsheet [1], we see that choosing a very low value for S_{LO} results in a wide range of required resistor value for A , B , and F . That is clearly not ideal in terms of relative accuracy of values. The best relative (ratio) accuracy will be when A , B , and F are close in value.

Applications

To give a real-world example, the author was involved in a smoke warning system. We initially considered an analog design based upon a metal oxide sensor (we definitely did not trust our code-writing skills to such an important application).

Suppose we need 3 seconds of smoke detection before the alarm output is activated, but 6 seconds without smoke before the alarm is silenced. **Figure 3** shows a possible circuit, **Figure 4** shows the simulated output.

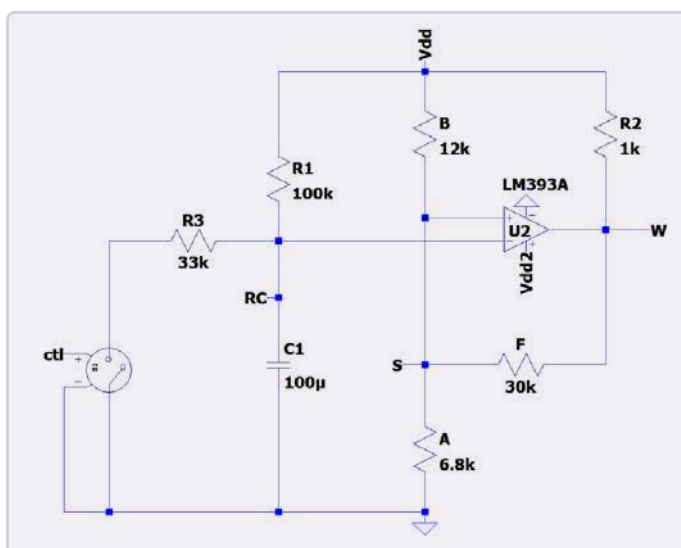


Figure 3: The input stage of a smoke detector circuit. When the switch is closed, $R_{EQ} = 25 \text{ k}\Omega$, $V_{EQ} = 1.25 \text{ V}$. When the switch is open, $R_{EQ} = 100 \text{ k}\Omega$, $V_{EQ} = 5 \text{ V}$. $\Delta V = (5 - 1.25) = 3.75 \text{ V}$.

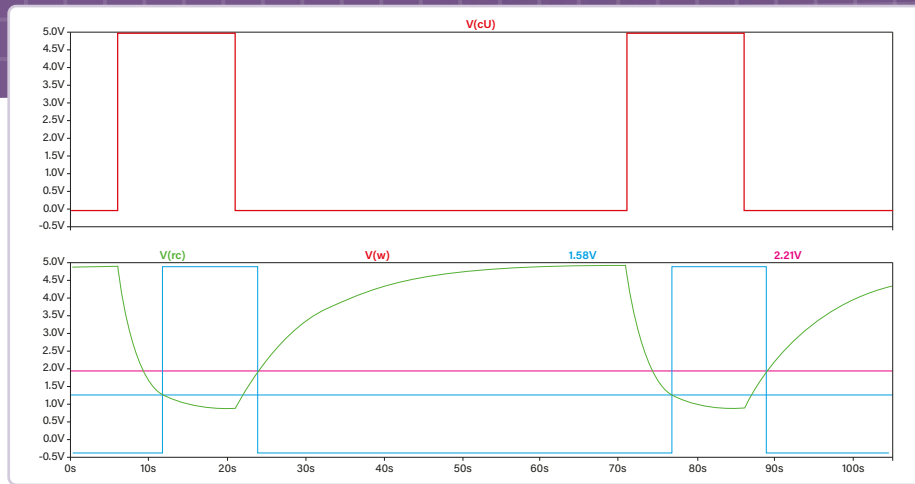


Figure 4: The simulated output of the circuit from Figure 3.

When no smoke is detected, the switch is on and the capacitor charges towards a baseline of 5/4 V through an effective resistance of $R_{EQ} = R1||R3$. When smoke is detected, the signal ct opens switch $S1$, causing the RC circuit to charge upward from 1.25 V toward $V_{dd} = 5$ V.

A spreadsheet gives the required trip-points, and another spreadsheet shows the conversions of those voltages into resistor values for A , B , and F .

Positive (Non-Inverting) Input Circuit

Now, let's look at the positive or non-inverting configuration, as shown in **Figure 5**. Such a circuit features an input impedance of $A+F$ and is a bit trickier to analyze.

Let:

$$\Delta S = S_{HI} - S_{LO}; \quad \Delta W = W_{HI} - W_{LO}$$

$$\Sigma S = S_{HI} + S_{LO}; \quad \Sigma W = W_{HI} + W_{LO}$$

Then:

$$\frac{A}{F} = \frac{\Delta S}{\Delta W}$$

$$M = \frac{A \cdot \Sigma W + F \cdot \Sigma S}{2 \cdot (A + F)}$$

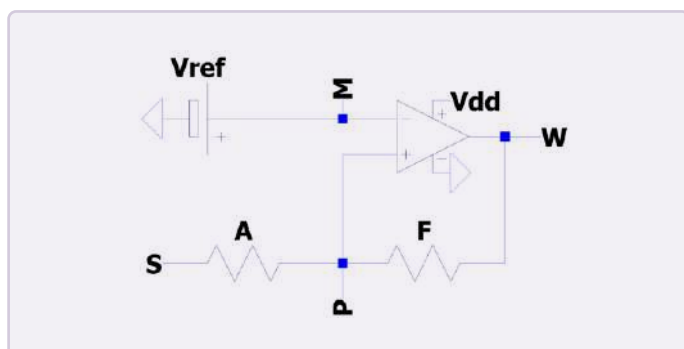


Figure 5: The positive or non-inverting configuration.

M can also be expressed independently of A or F :

$$M = \left(\frac{1}{2} \right) \cdot \left(\frac{1}{\Delta W + \Delta S} \right) \cdot (\Delta S \cdot \Sigma W + \Delta W \cdot \Sigma S)$$

Example

Suppose the lower trip value $S_{LO} = 0.5$, the upper trip value $S_{HI} = 3.0$, while the output levels are $W_{LO} = 0$ and $W_{HI} = 5$. Then, $A/F = (3-0.5)/(5-0) = 0.5$ and $M = 0.5 \times 1/(5+2.5) \times (2.5 \times 5 + 5 \times 3.5) = 2$ V.

Figure 6 shows the results.

Current-Input Hysteresis

Connecting resistor A to ground transforms the circuit from Figure 3 into one suitable for a current-signal-type input, as shown in **Figure 7** and **Figure 8**. The value of resistor F is found directly from ΔW and ΔI . Then, divider ratio β can be found. Finally, the value of A is calculated. A spreadsheet helps here because both β and A should be as accurate as possible using standard resistor values.

$$\beta = \frac{A}{A + F}; \quad R_p = \frac{A \cdot F}{A + F}$$

$$P = \beta \cdot W_H + I_L \cdot R_p = \beta \cdot W_L + I_H \cdot R_p$$

$$\beta \cdot \Delta W = R_p \cdot \Delta I; \quad \frac{R_p}{\beta} = F = \frac{\Delta W}{\Delta I}$$

$$W_L = \frac{P - I_H \cdot R_p}{\beta} = \frac{P - I_H \cdot F \cdot \beta}{\beta} = \frac{P}{\beta} - I_H \cdot F$$

$$\frac{P}{\beta} = W_L + I_H \cdot F \rightarrow \beta = \frac{P}{W_L + I_H \cdot F}$$

$$A = \frac{F \cdot \beta}{1 - \beta}$$

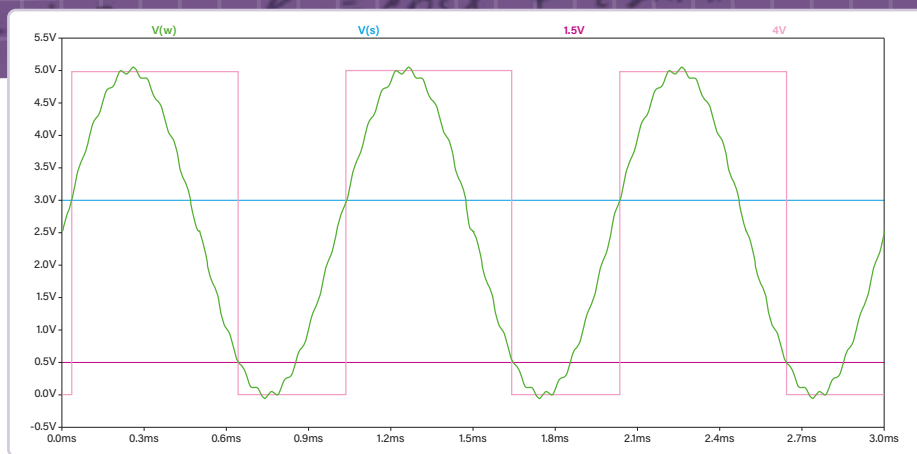


Figure 6: The simulated signals of the circuit from Figure 5.

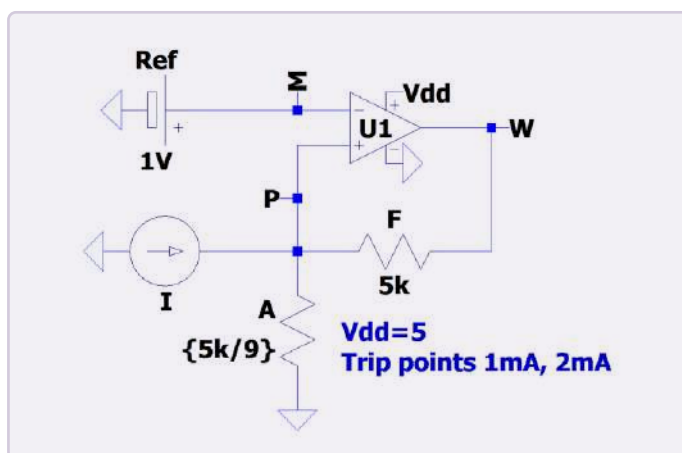


Figure 7: The circuit in Figure 3 adapted for a current input.

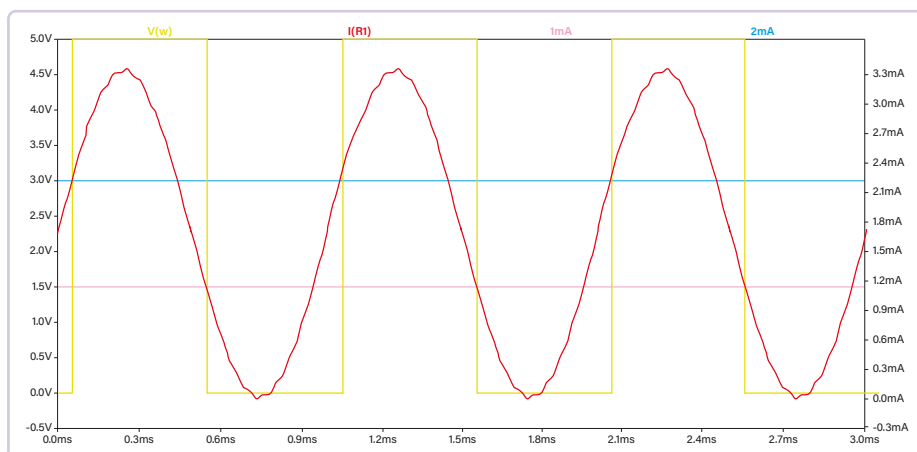


Figure 8: The simulator output for the circuit from Figure 7.

Simulations, Spreadsheets, and Algebra

Simulations, spreadsheets, and full derivations for the formulae given in this article are available for download from this article's webpage [1].

200559-01

Questions or Comments?

Do you have technical questions or comments about this article? Contact Elektor at editor@elektor.com.



Related Products

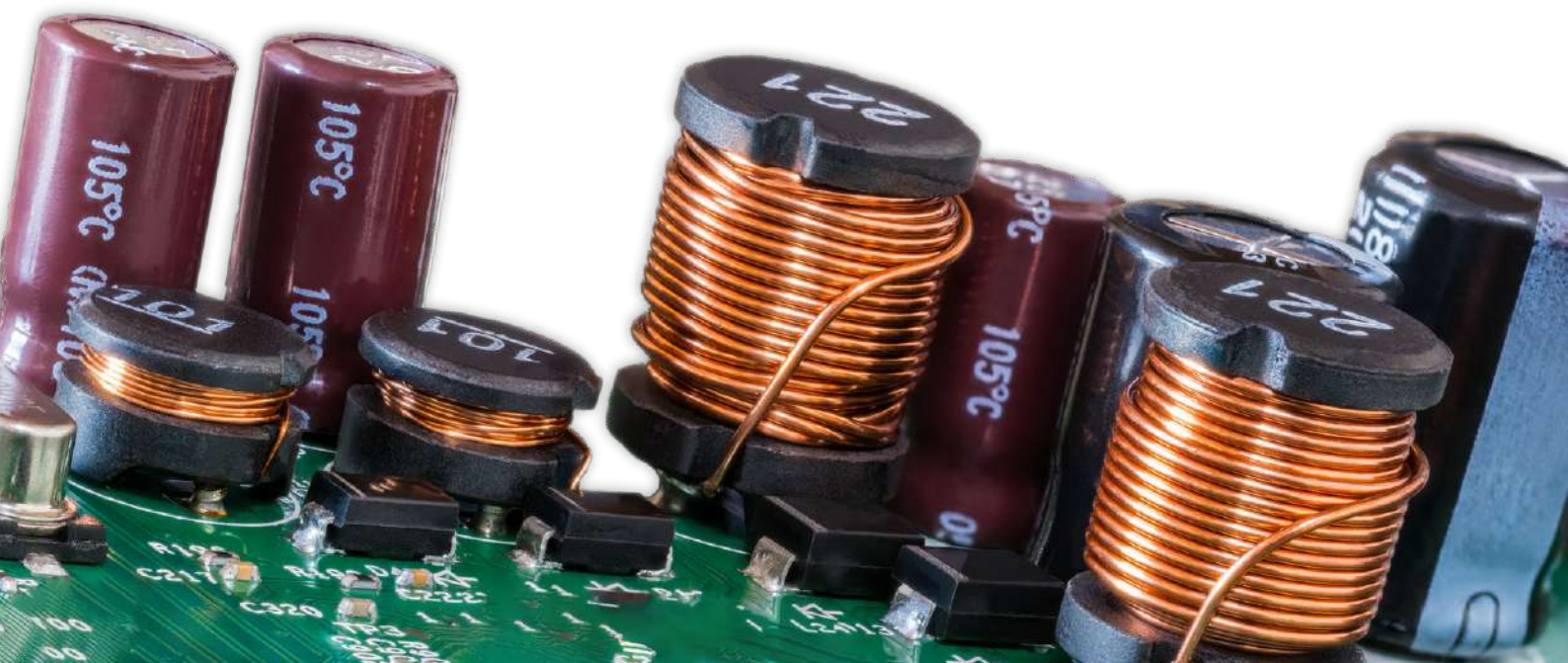
- > Paul Horowitz, Winfield Hill, *The Art of Electronics (3rd Edition)* (SKU 17167) <https://elektor.com/17167>
- > Gilles Brocard, *The LTspice XVII Simulator* (SKU 19741) <https://elektor.com/19741>

WEB LINK

[1] Downloads for this article: <https://elektormagazine.com/200559-01>

ESP32-Based Impedance Analyzer

Simple, Low-Part-Count, and Inexpensive!



By Volker Ziemann (Sweden)

An ESP32 as an impedance analyzer that can even check the resonance of an LC network? Yes, this inexpensive controller board can do the job, with just a handful of external components, and yet offers a web-based user interface.

Let's start with some theory. The impedance of an electronic component $Z = U/I$ is the ratio of the voltage U dropped across the component and the current I flowing through it. Ohm's law, written in the form of $R = U/I$, is a special example where the resistance R is the simplest form of an impedance. For a capacitor with capacitance C , the impedance depends on the frequency $\omega = 2\pi f$ and is given by $-i/\omega C$; the imaginary unit i is a compact way of indicating that the phase of the voltage lags that of the current by 90° . Likewise, the impedance of an inductor with inductance L is given by $i\omega L$. Again, the imaginary unit i indicates now the voltage leads the current. We can thus figure out the impedance of a component by probing the frequency dependence of its impedance and the phase relation between voltage and current.

These measurements get especially interesting with more than one component. If a coil and a capacitor are connected in parallel, the circuit has the highest impedance at the resonant frequency at which the two AC resistances are equal. With a series connection, the impedance is lowest at this point. With the Espressif ESP32-based Impedance Analyzer, one can now record the frequency-dependent impedance curve of RLC networks and thus determine their characteristics.



All we need is a way to produce a sine wave signal with constant amplitude, sweep its frequency in a preferably wide range, and then rapidly measure the current that flows through the component(s), generically referred to as device under test (DUT). Remarkably, the ESP32 microcontroller that I received from Elektor to participate in a 2018 design contest can do most of these things. It has a built-in frequency generator and a DAC that outputs voltages with frequencies up to a few 100 kHz. Moreover, it features an ADC that reads voltages at speeds of up to a million times per second, though with a bit of cheating; but we'll get to that. Since the ESP32 can only read one ADC channel at high speed, and the ADC and DAC run independently, we can only determine the magnitude of the impedance, but not the phase, with the ESP32 alone. The access to the phase shift caused by the DUT needs special hardware, and therefore it will be the topic of a separate article.

Analog Front End

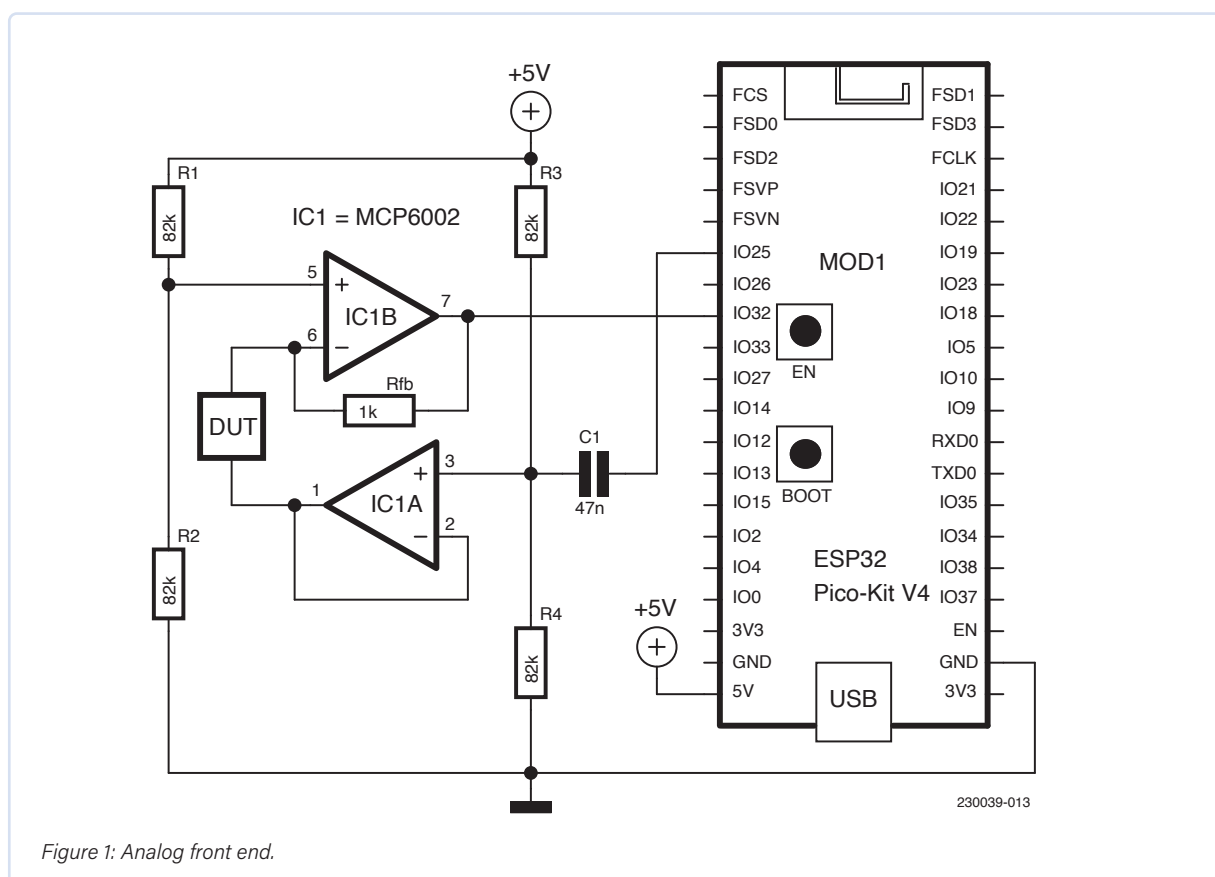
We must ensure that the amplitude of the DAC voltage is constant and does not depend on the impedance of the DUT. That is the task of the lower opamp in **Figure 1**. It serves as a 1x buffer for the signal from pin 25 of the ESP32. Its low impedance output drives the DUT. The other terminal of the DUT is connected to the negative input of a second opamp, which is wired as a transimpedance amplifier. A current flowing into its negative input terminal is converted with the

help of the feedback resistor R_{fb} to a voltage on its output, which is routed to the ADC pin 32 of the ESP32. This ADC can only handle positive voltages. Therefore, the DC level of the signal is shifted to half the supply voltage with $R1$ and $R2$. With this circuit, the DUT is fed with an alternating voltage of constant amplitude while we measure the current with the ADC.

Figure 2 shows the analog front end assembled on a small veroboard in the front with the capacitor as DUT shown on the lower left. The resistor mounted upright next to the opamp is the easy-to-change R_{fb} . The four wires connecting this small board with the ESP32 are GND (black), 3.3 V (red), DAC pin 25 (green), and ADC pin 32 (blue).

High-Speed ADC

The high-speed mode of the ADC on the ESP32 is part of the I2S subsystem that is normally used to manipulate digital sound. On top of this, the ESP32 supports a mode where output words from the ADC are copied into a buffer using direct memory access (DMA) and therefore do not require the CPU. The ADC is free running, and DMA grabs data at a specified rate. As long as this rate is lower than the maximum conversion rate of the ADC, which is about 250 kS/s, all samples are "fresh." On the other hand, if the rate is too high, the same samples are copied into the buffer, which show up as staircases in the data of the captured waveform.



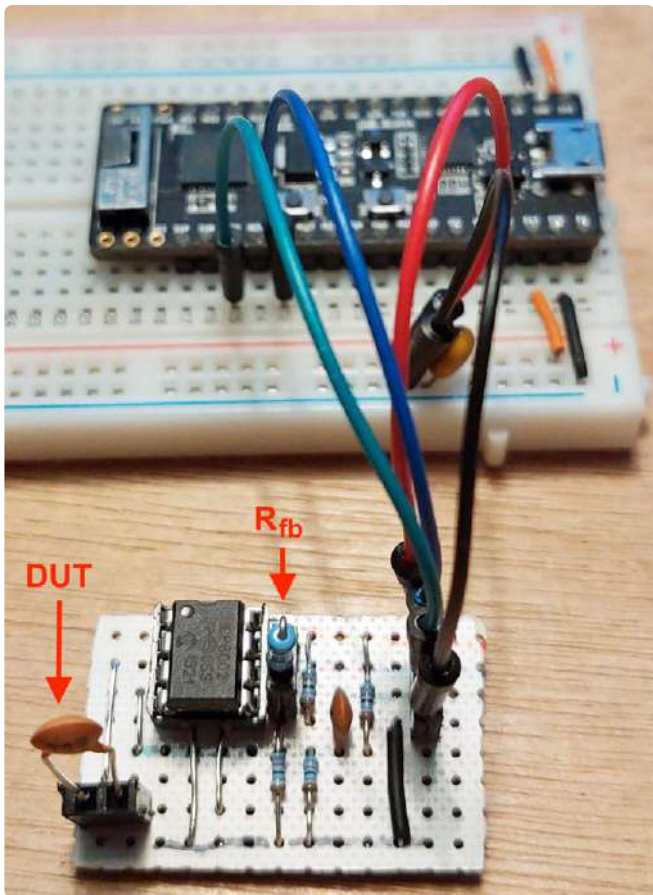


Figure 2: To let the ESP32 measure the impedance of a DUT, we need just a few external components.

The ADC sampling is set by the function `is2Init()`, which is derived from the sketch `HiFreq_ADC.ino` that is part of the ESP32 support package for the Arduino IDE. Most of the configuration consists of filling the structure `is2_config` with the desired mode of operation, the sampling rate, and a few flags that were found after quite some archaeology in [1]. Before leaving the function, we set the attenuation level such that the full scale of the ADC corresponds to the supply voltage of 3.3 V, choose the ADC channel, and enable the output.

Frequency Generation

The ESP32 has a built-in functional block that spits out successive values of a cosine function at an adjustable rate. The output from this block can be routed, by suitably setting several configuration bits, to the input register of the DAC, which results in a cosine-like output voltage. This frequency generator is controlled by directly writing to the registers `SENS_SAR_DAC_CTRL1_REG` and `SENS_SAR_DAC_CTRL2_REG`, which is described in [1] and, more detailed, in [2]. Following the explanations from [2], the function `cwDACinit()` is prepared to fill these registers with values to configure the output frequency and amplitude. After including header files that enable us to use mnemonic names, we use `SET_PERI_REG_MASK(reg,bits)` to set `bits` in register `reg`. For example, the first of the following commands

```
SET_PERI_REG_MASK(SENS_SAR_DAC_CTRL1_REG,
    SENS_SW_TONE_EN);
SET_PERI_REG_BITS(SENS_SAR_DAC_CTRL1_REG, SENS_SW_FSTEP,
    frequency_step, SENS_SW_FSTEP_S);
```

enables the internal high-speed frequency generator. The second determines the output frequency by specifying the integer `frequency_step`, whose value is derived from an internal clock. The desired output frequency follows a slightly adjusted equation taken from [1].

ESP32 Sketch, Controlled Via the Serial Line

I programmed the ESP32 with the legacy version 1.8.19 of the Arduino IDE, available from [3], and followed the instructions to install it. At the time of writing, the new version 2 of the IDE did not yet support an add-on needed later. I also had to install the ESP32 support functions from [4] and to follow the installation instructions.

The following sketches are based on [5], where an in-depth discussion of many aspects can be found. The intention is to control the frequency generation and the acquisition from the serial line based on a simple protocol where only strings are sent back and forth between the ESP32 and a program on a PC that can handle serial communication; this includes Python, Octave, and LabView. This protocol resembles the SCPI protocol used in oscilloscopes and other measurement equipment, where an appended question mark indicates a command that expects a reply from the ESP32. Sending `STATE?`, for example, triggers the reply `STATE fmin fmax fstep`, where the three numerical values identify the range and the step size of the frequency sweep.

Now, the different parts of the sketch. First, two header files are included — one for the home-grown support functions for ADC and DAC, the other to access a flash-based SPIFFS file system on the ESP32. The latter serves to store calibration data. Default values for the range of the frequency sweep and other pertinent variables are specified also. The `setup()` function of the sketch initializes the serial communication, sets the output frequency and amplitude of the DAC, and reads calibration data from the SPIFFS file system if it is available.

The function `loop()` checks whether some request has arrived from the host computer with the call to `Serial.available()`, reads one line (terminated by a line feed `\n` character), and converts whatever has arrived to the character array `line`. Now, some tests are performed whether the line starts with a specific string. For example, if `line` contains `FREQ 20000`, the numerical value is extracted with `atoi(&line[5])`. It converts the string that starts at the fifth position in `line` to an integer with the built-in function `atoi()`. Then it copies that value to the variable `dac25freq` and initializes the frequency generator with `cwDACinit()`. Similarly, all relevant variables are accessible from the host computer.

The most interesting command is `SWEEP?`. After receiving this command and declaring the variables used in this section, a for-loop iterates the variable `f` from `fmin` to `fmax` with a step size of `fstep`. Inside the loop, first the frequency of the DAC to this frequency is set, and after a short delay, the measurements from the ADC are retrieved with `is2_read()`.

```
for (float f=freqmin; f<freqmax;f+=freqstep) {
    dac25freq=f;
    cwDACinit(dac25freq,dac25scale,0);
    delay(50);
```



```
i2s_read(I2S_NUM_0, &buffer,  
        sizeof(buffer), &bytes_read, 15);  
...  
}
```

The size of its input argument `buffer`, defined near the top of the sketch, is used to determine the number of samples, here 1024. The function also returns the number of `bytes_read`. Since each sample uses two bytes, we need to divide by two when looping over the samples to determine the minimum and maximum values. Remembering that the voltage measured by the ADC is proportional to the current flowing through the DUT, we use this peak-to-peak variation as a measure of the amplitude of the current.

Calibration

While looping over the samples, the sums `q1` and `q2` are also accumulated. Using them jointly with the variables `S0`, `S1`, and `S2` fits a straight line to the data points, whose parameters are needed for the calibration. The details of the algorithm are explained in Appendix B of [5]. The command `SAVECALIB` saves these parameters to the persistent SPIFFS memory. The command `GETCALIB?` retrieves them from the PC.

The calibration constants `calib_slope` and `calib_offset` are needed to account for various unknown attenuation factors in the system, for example, due to the AC coupling capacitor near the input to the lower opamp. This ambiguity is resolved by calibrating the system with a resistor of known resistance inserted as DUT. All other impedances are then determined with respect to this calibration resistor. Ideally, this resistance should be independent of the frequency, but small systematic changes are taken into account by using the slope as a function of frequency in addition to the offset. The value of the calibration resistor should have roughly the same value as the feedback resistor of the transimpedance amplifier to match the range of the ADC.

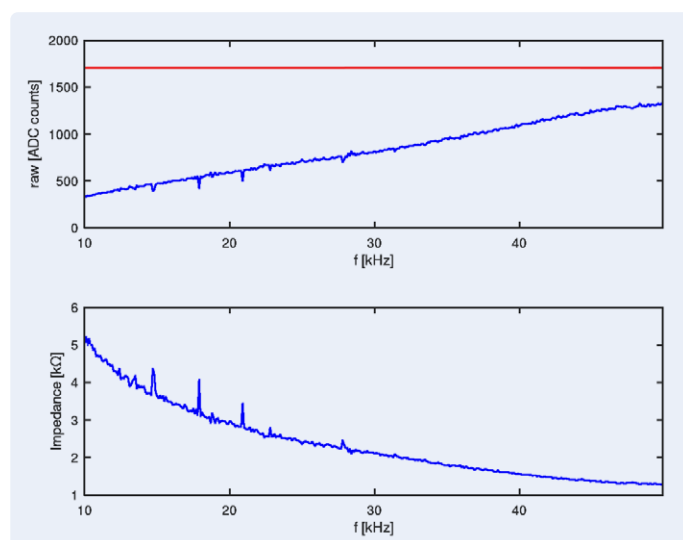


Figure 3: Raw readings and impedance for a capacitor as DUT.

Before using the system, the SPIFFS file system on the ESP32 should be initialized by creating an empty directory named `data` inside the directory where the sketch is stored. Then, the Arduino ESP32 filesystem uploader plugin from [6] needs to be installed following the instructions. After that is done, a new menu point *ESP32 Sketch Data Upload* appears in the *Tools* menu of the Arduino IDE. After making sure that the serial monitor is closed, clicking on that entry will create the SPIFFS file system on the ESP32.

Control From Octave or Python

Controlling the data acquisition from the PC only involves opening the serial port, whose name differs between operating systems. It is typically called `COMn` on Windows, `/dev/tty.usbserial-n` on a MAC, or `/dev/ttyUSBn` on Linux. Sending a command, for example to set the starting frequency of the sweep only involves writing `FMIN 10000` to the serial port. Likewise, reading the current settings involves writing `STATE?`, waiting a little while, and reading the characters that come back, up to a line feed `\n` character. The response reads `STATE fmin fmax fstep`. Performing a frequency sweep is initiated by writing `SWEEP?` and receiving the values, one value per line at a time. Once all data is available, we can close the serial port and prepare a plot.

Scripts, both for Octave with installed instrument toolbox and for Python with the *python-serial* package are included in the software archive accompanying this article. Note that for initial tests, even a terminal program such as Putty can be used.

Using the System

To start using the system, a 1 kΩ calibration resistor is used as DUT and a frequency sweep is performed. Subsequently issuing the command `SAVECALIB` which is by default commented out in the accompanying Octave script `nwa.m`, stores the calibration data on the ESP32. If now the `SAVECALIB` command is commented out again, the resistor

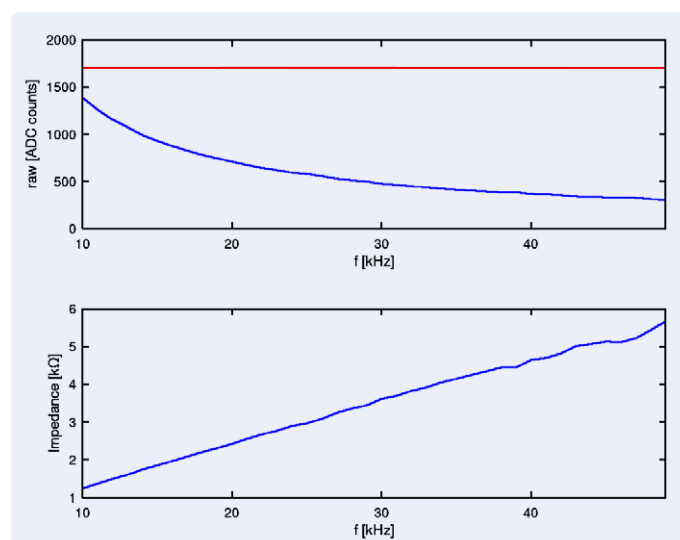


Figure 4: Raw readings and impedance for an inductor as DUT.

is replaced by a 2.2 nF capacitor, and `nwa.m` is executed, the properly calibrated plot shown in **Figure 3** appears. It shows the expected inverse dependence on the frequency from which we determine the capacitance with the Octave command:

```
capacitance_nF=
    mean(1./(2*pi*Zabs*1e3.*xx*1e3))*1e9
```

with the formula:

$$C = 1/(2\pi f Z_{abs})$$

Averaging over all available data points is done with the built-in function `mean()`. The powers of ten are needed to account for the kilos in kΩ and kHz, as well as the nanos in nF. For example, `1e9` at the end converts the capacitance from Farad to nanofarad.

Repeating the sweep with the capacitor replaced by a 22-mH inductor, results in the plots shown in **Figure 4**. As expected, the impedance of an inductor, shown on the lower panel, rises linearly with the frequency, so the inductance can be determined from formula:

$$L = Z_{abs}/2\pi f$$

or written in Octave, from:

```
inductance_mH=
    mean(1e3*Zabs./(2*pi*xx*1e3))*1e3
```

which also averages over all data points with `mean()`. Again, the powers of ten are needed to account for the kilos in kΩ and kHz, as well as the millis in mH.

Web-Based User Interface

So far, the impedance measurement is controlled from Octave or Python, but now a web browser is used to control and display the measurements. Therefore, the ESP32 is configured to run a web server that publishes the web page shown in **Figure 5**. Once the browser receives this web page, embedded JavaScript code opens a second communication channel, based on *websockets*, to the ESP32. Here, a websocket assumes the role of the serial line and is used to send text-based messages back and forth. Sending these messages is triggered by clicking on the buttons shown near the top of the web page. They initiate a frequency sweep, clear the displayed traces, and save the calibration data. On the line below, parameters controlling the ESP32, including the range of the sweep and the sample rate of the ADC can be adjusted from selection menus. The buttons on the third line calculate the capacitance, resistance, and inductance and display the result on the status lines below the graph. The lower line displays information received from the ESP32. On the graph, the horizontal axis shows the frequency axis in the range specified in the menus near the top of the page. The vertical axis shows Z_{abs} relative to the calibration resistor, which is displayed as the horizontal blue line. Clicking on the graph displays the frequency and Z_{abs} in the status line.

Sketch on the ESP32

After adapting the Wi-Fi network name (SSID) and its password, the support files for Wi-Fi, websocket and webserver need to be included, then we configure the web server `server2` to listen on network port 80, and the `websocket` to communicate via port 81. The text-messages sent between browsers are JSON-formatted, having the form `{"INFO": "Yada yada"}`. Parsing these messages and creating more complex messages is handled by the `ArduinoJson` library, whereas support for the ADC and DAC is provided by `ESP32_I2Sconfig.h`, as before.

```
const char* ssid      = "YOUR_SSID";
const char* password  = "YOUR_PASSWORD";
#include <WiFi.h>
#include <WebSocketsServer.h>
WebSocketsServer webSocket = WebSocketsServer(81);
#include <WebServer.h>
#include <SPIFFS.h>
WebServer server2(80);
#include <ArduinoJson.h>
#include "ESP32_I2Sconfig.h"
```

After specifying several variables, helper functions are defined, of which `websocketEvent()` is the most important. It is called whenever a message from the browser arrives. If it is a JSON-formatted message, the following code snippet extracts the command `cmd` and value `val` with the help of functions from the `ArduinoJson` library. Depending on the command, it then triggers the appropriate actions. For example, receiving the command `SWEEP`, it reports back to the browser with `sendMSG()` and sets the variable `mmode` to one, which is interpreted in the main loop. Since `websocketEvent()` is called asynchronously it interrupts other activities and this should be kept as short as possible; therefore, the sweep is deferred to the main program.

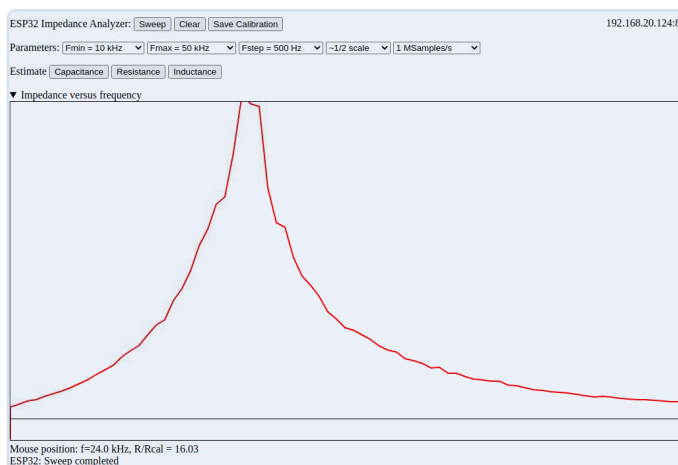


Figure 5: The ESP32 produced web page shows a resonance in the graph with the impedance of a 2.2-nF capacitor, 22-mH inductor, and 33-kΩ resistor connected in parallel.



On the other hand, setting the start frequency of the sweep `freqmin` does not use many CPU cycles and is handled inside `WebSocketEvent()`.

```
DynamicJsonDocument root(300);
deserializeJson(root,payload);
const char *cmd = root["cmd"];
const long val = root["val"];
if (strstr(cmd,"SWEEP")) {
    sendMSG("INFO","ESP32: Received Sweep command");
    mmode=1;
} else if (strstr(cmd,"FMIN")) {
    freqmin=val;
    Serial.printf("FreqMin = %g\n",freqmin);
} else
...

```

A list of commands that resemble those used earlier is handled similarly.

The following code snippet from the `setup()` function first connects to the WLAN with the credentials provided before. It prints dots to the serial line until it succeeds, and then prints the IP number, starts the websocket communication, and registers `WebSocketEvent` to handle messages arriving via websocket. Note that the serial line is not strictly needed in this part, but it is nevertheless comforting to observe what is happening on ESP32, especially while developing the system.

```
WiFi.begin(ssid, password);
while(WiFi.status() != WL_CONNECTED)
    {Serial.print("."); delay(500);}
Serial.print("\nConnected to ");
Serial.print(ssid);
Serial.print(" with IP address: ");
Serial.println(WiFi.localIP());
WebSocket.begin();
WebSocket.onEvent(WebSocketEvent);

```

Then, still in `setup()`, we check that the SPIFFS file system exists, just like in the first part, and load the calibration data from it. Now, however, the SPIFFS filesystem also contains the file `esp32_impedance_analyzer.html`, which describes the web page. After starting `server2`, this HTML file is registered to be delivered to connecting browsers per default, as indicated by the first argument in `server2.serveStatic()`.

```
server2.begin();
server2.serveStatic("/",SPIFFS,
    "/esp32_impedance_analyzer.html");

```

The remainder of the `setup()` function largely resembles the earlier sketch.

In the function `loop()` first anything related to the http and websocket servers is handled, before the variable `info_available` is checked. This is set by `sendMSG()` to announce that the `info_buffer` contains a

JSON-formatted message, which is promptly dispatched to the browser with a call of `WebSocket.sendTXT()`.

```
server2.handleClient(); // handle http server
WebSocket.loop();       // handle websocket server
if (info_available==1) {
    info_available=0;
    WebSocket.sendTXT(websock_num,
        info_buffer,strlen(info_buffer));
}

```

Then the value of `mmode` is checked, and the appropriate action is started. For example, `mmode==1` performs the frequency sweep with code that is very similar to that used before. Here, only a JSON-formatted message with the name `WF0` (for waveform) is built and saved in the variable `doc` with:

```
doc["WF0"][nn-1]=floor((512/16)*Z);

```

where the variable `nn` loops over the frequency points and `Z` is the absolute value of the impedance at that frequency. Note that the variable is scaled for 512 pixels covering the range from zero to 16 kΩ, and the values are converted to an integer with `floor()`. After the loop has finished, the waveform is dispatched to the browser with:

```
serializeJson(doc,out);
WebSocket.sendTXT(websock_num,out,strlen(out));
sendMSG("INFO","ESP32: Sweep completed");

```

and announces that the sweep has finished with `sendMSG()`. The other values of `mmode` save the calibration and report the estimated capacitance, resistance, or inductance to the browser.

The file `esp32_impedance_analyzer.html` describes the webpage and contains the JavaScript that brings it to life.

The Webpage

Most interactive web pages use `<style>` tags to describe generic aspects of the appearance of things that show up on the page, followed by HTML commands that describe the webpage, and JavaScript instructions to make it interactive.

The following style instruction near the top of `esp32_impedance_analyzer.html` cause a border drawn around the display area and that two displayed traces are shown in red and black. The last instruction causes the IP number to be displayed on the right-hand side of the webpage:

```
<style>
#displayarea { border: 1px solid black; }
#trace0 { fill: none; stroke: red; stroke-width: 2px;}
#trace1 { fill: none; stroke: black; stroke-width: 1px;}
#ip {float: right;}
</style>

```

The main part of the web-page description is enclosed between the tags `<BODY>` and `</BODY>`. Right near the top is the description of the buttons. The definition of the first button is enclosed between `button` tags and reads:

```
<button id="sweep" type="button"
  onclick="sweep();">Sweep</button>
```

It displays `Sweep` on the button, and a click executes the JavaScript function `sweep()`. Such actions tied to an event are commonly referred to as *callback functions*. Assigning an ID, which is `sweep`, to the button allows us to later change its properties, for example, the displayed text or which action to trigger. Defining the other buttons follows the same scheme.

The selection menu on the second line follows a slightly different syntax. The definition of this menu is enclosed between `SELECT` tags. If one of the entries is selected, it calls `setDacFreqMin(this.value)`, where `this.value` is the value specified in the different `OPTION` tags. The `OPTGROUP` tags are only ornamental.

```
<SELECT onchange="setDacFreqMin(this.value);">
<OPTGROUP label="Sweep start frequency">
<OPTION value="1000">Fmin = 1 kHz</OPTION>
<OPTION value="2000">Fmin = 2 kHz</OPTION>
<OPTION value="5000">Fmin = 5 kHz</OPTION>
<OPTION selected="selected" value="10000">
  Fmin = 10 kHz</OPTION>
<OPTION value="20000">Fmin = 20 kHz</OPTION>
<OPTION value="50000">Fmin = 50 kHz</OPTION>
</OPTGROUP>
</SELECT>
```

Finally, the area to display the impedance is a *Scalable Vector Graphic* (SVG) that has the specified size of 1024 × 512 pixels and displays two `path` with ID `trace0` and `trace1`. The `id` later allows us to modify them. The property `d` describes the waveform. Initializing is done by moving the cursor to pixel (0,200) with `M0 200`. Later, `d` is redefined with the waveform `WF0` that arrives from the ESP32.

```
<svg id="displayarea" width="1024px" height="512px">
<path id="trace0" d="M0 200" />
<path id="trace1" d="M0 200" />
</svg>
```

Finally, the two status lines are defined with:

```
<div id="status">Status window</div>
<div id="reply">Reply from ESP32</div>
```

They are named via `id`, which later allows to change the displayed text. Likewise, the area to display the IP number is named on the top right `ip`.

JavaScript

All JavaScript commands are enclosed by `SCRIPT` tags. Following the opening tag several variables are defined, and the IP number of the ESP32 is determined with:

```
var ipaddr=location.hostname + ":81";
document.getElementById('ip').innerHTML=ipaddr;
```

Here the port number of the websocket is appended on the ESP32 and immediately the text of the tag, labeled `ip`, is updated. The space to display text is accessed by the somewhat lengthy construction in the second line, where `document` refers to the webpage itself. The part after the period gives access to the named element `ip`, and `innerHTML` refers to the displayed text that is changed to show `ipaddr`. This construction is used extensively to access named elements and to change their properties. The functions `toStatus()` and `toReply()` follow this template to copy text to the status lines at the bottom of the webpage.

Knowing the address of the websocket on the ESP32, it is opened with `new WebSocket()` and then callback functions are attached to the events `onopen`, `onclose`, and several others. Often just a short message is sent to the JavaScript console with `console.log()`. The console is accessible from the browser's *Developer tools* or with `Ctrl-Shift-I` on Chromium.

```
var websock = new WebSocket('ws://' + ipaddr);
websock.onopen = function(evt)
  {console.log('websock open')};
```

The most interesting callback function reacts to arriving messages from the ESP32. The slightly shortened function `websock.onmessage()` first parses the arriving JSON-formatted message, which is stored in `event.data` and places the command-value pairs into the structure `stuff`. If `stuff` contains the command `INFO`, the associated value is stored in `val` and displayed on the webpage with `toReply()`. If the command is `WF0` it contains the waveform with the impedance data. The number of arriving data points is determined with `val.length` and used to match the horizontal axis to fill the available 1024 pixels. Then the property `d` of the path is initialized, and points are appended to it, one for each entry in `val`. Since pixel (0,0) is at the top left, vertical position needs to be inverted by setting pixel `512-val[i]`. Finally, the black reference line with the calibration resistance will be displayed.

```
websock.onmessage=function(event) {
var stuff=JSON.parse(event.data);
var val=stuff["INFO"]; // info
if (val != undefined) {toReply(val);}
var val=stuff["WF0"]; // waveform0
if (val != undefined) {
  nstep=Math.floor(0.5+1024/val.length)
  pixmax=nstep*val.length;
  var d="M0 511";
  for (i=0; i<val.length; i++)
```



```
{d += ' L' + (nstep*i) + ' ' + (512-val[i]);}
document.getElementById('trace0').setAttribute('d',d);
d="M0 480 L1024 480";
document.getElementById('trace1').setAttribute('d',d);
}
}
```

The remainder of the JavaScript is mostly callback functions to buttons and menus. The function `sweep()`, triggered by the corresponding button, reads:

```
function sweep() {
websock.send(JSON.stringify
    ({ "cmd" : "SWEEP", "val" : -1 }));
}
```

The function `JSON.stringify()` packs its argument into a properly formatted message and `websocket.send()` dispatches it to the ESP32. All the other callback functions follow the same template.

Just before the end of the JavaScript section, `showCoordinates()` is defined to display the frequency and impedance that corresponds to the pixel on the graph area that is clicked. This function gets a callback of a `mousedown` event by attaching an `addEventListener()` to `displayarea`:

```
document.getElementById("displayarea")
    .addEventListener('mousedown', showCoordinates, false);
```

This feature makes it rather convenient to directly determine frequencies and the corresponding impedances from the displayed graph.

In Figure 5 we can see that the resonant behavior of a parallel RLC circuit we used as DUT shows a resonance peak near 24 kHz, where the impedance exceeds 16 kΩ.

The firmware and others files can be downloaded for free from [7]. Now the impedance analyzer is self-sufficient in the sense that no controlling program is required. All communication happens between the ESP32 and a browser, even one running on a smartphone. ◀

230039-01

Questions or Comments?

Do you have technical questions or comments about this article? Email Elektor at editor@elektor.com.

About the Author

Volker Ziemann's interest in electronics started around the 40 W Edwin amplifier (Elektor, mid-70s), but he got sidetracked and studied physics and worked with particle accelerators ever since – at SLAC in the US, CERN in Geneva, and now in Uppsala, Sweden. Since electronics plays an important role for the control and data acquisition in accelerators, his early interest was useful throughout his career. He now teaches at Uppsala University. One of his books about data acquisition with Arduino and Raspberry Pi touches the topic of this article.



Related Products

➤ **ESP32-DevKitC-32D (SKU 18701)**
www.elektor.com/18701

➤ **OWON HDS1021M-N 1-ch Oscilloscope (20 MHz) + Multimeter (SKU 18778)**
www.elektor.com/18778

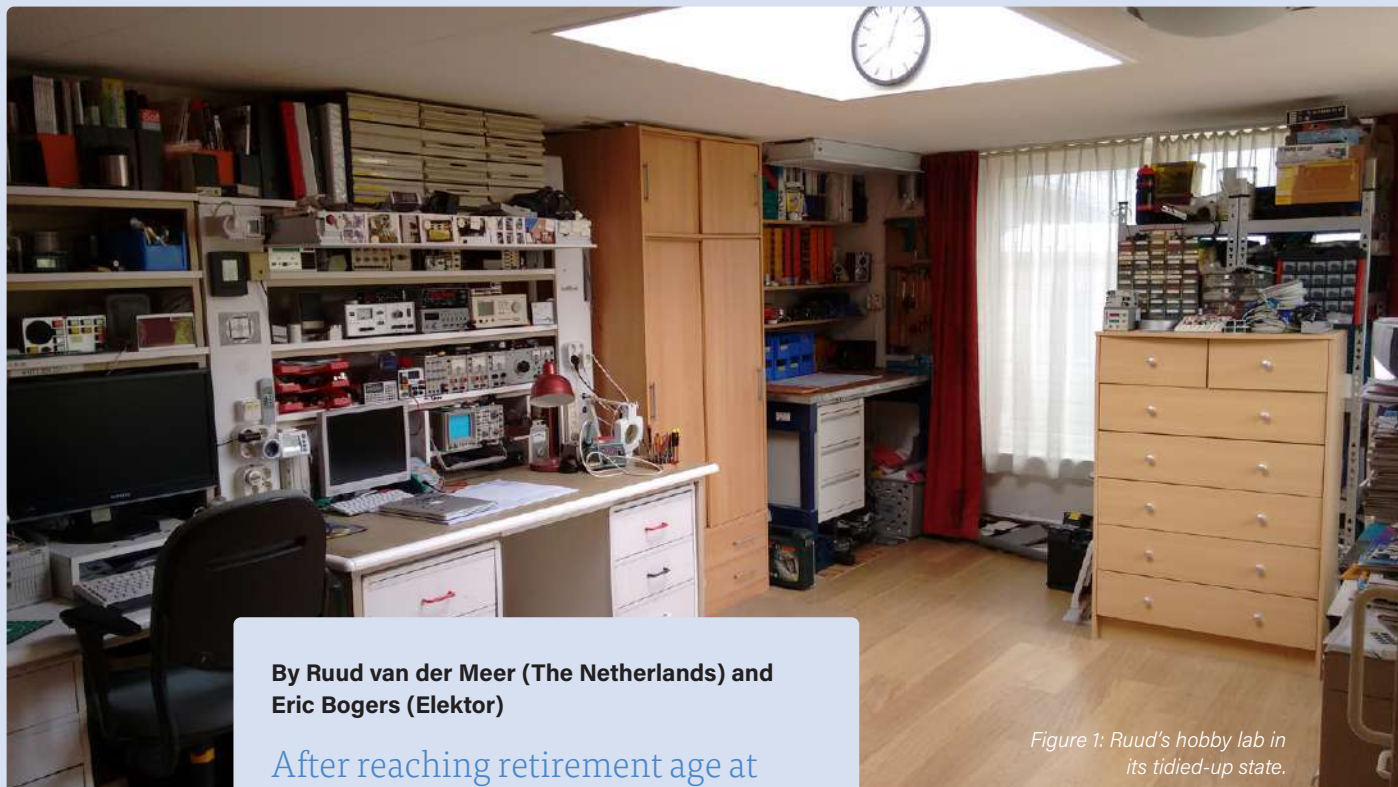
WEB LINKS

- [1] ESP32 Technical Reference Manual (Version 4.7), available from : https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf
- [2] ESP32 cosine waveform generator explained: <https://github.com/krzychb/dac-cosine>
- [3] Arduino website: <https://www.arduino.cc>
- [4] Arduino Support Functions: <https://github.com/espressif/arduino-esp32>
- [5] V. Ziemann, A Hands-on Course in Sensors Using the Arduino and Raspberry Pi, 2nd ed., CRC Press, Boca Raton, 2023:
- [6] ESP32fs Plugin: <https://github.com/me-no-dev/arduino-esp32fs-plugin>
- [7] Source code for this project at GitHub: <https://tinyurl.com/4awvvec>



HomeLab Tours

Encouraging DIY



By Ruud van der Meer (The Netherlands) and Eric Bogers (Elektor)

After reaching retirement age at the end of a long and successful career in electronics, many people devote their attention to something completely different – such as growing tomatoes or breeding guinea pigs – as long as it has nothing to do with electronics. Ruud van der Meer, who hails from Roelofarendsveen, is one of the exceptions. After more than 40 years at Siemens, his enthusiasm is still unstoppable.

After completing vocational school and a course of study in measurement and control technology plus advanced training as an electronics technician, Ruud started working at Siemens in the department for measurement and control equipment, measuring instruments, and calibration in 1970. In Ruud's own words:

“There were lots of opportunities. In just a short time, I developed a number of projects, including LED displays, PI controllers and measurement transducers, and by developing test equipment and test systems I was able to improve a lot of work processes.

Figure 1: Ruud's hobby lab in its tidied-up state.

After several years, I became a team leader, and, together with the Siemens Hobby Computer Club, I developed our own PC system called SUMO80.”

Ruud quickly noticed that some people needed a bit of help to find their way with new technologies, so he started teaching courses inside the company and at the Regional Training Center in the fields of electrical engineering and electronics, digital methods, communications technology, and mechatronics.

“At a certain point, I was looking for a new challenge, and that came along when I was asked to be the head of company vocational training at Siemens. At the time, there were approximately 60 students in the electrical engineering program. One of my tasks (along with the theoretical part of the program) was to improve the inflow of new people in the company, which had declined over the years. Among other things, I did that by expanding the training program to include electronics, PLC technology, and communications technology, as well as mechatronics in the last year, in cooperation with lecturers at TU Delft.”

The Siemens company vocational training program was shut down in 2005, so Ruud had to look for a job in another field (but still within Siemens). In the end,



There's not enough DIY electronics being built now.



he found a new position in the department responsible for building automation and fire protection.

“To make a long story short, in terms of work processes, it looked like the Stone Age there, but, after a while, I managed to make the department more streamlined. And then I retired in 2016.”

It was again time for something different. Ruud has become the first energy coach of the municipality of Kaag en Braassem, and, in the meantime, has made a lot of recommendations regarding sustainability. In part due to this, he has developed many home automation systems based on Arduino.

“After a number of presentations to the Hobby Computer Club about Arduino, I was asked to do the same for the Leiderdorp Adult Education Center (LVU). That started six years ago. After that, I also taught Arduino courses at the adult education centers of Alphen aan de Rijn, Lisse, and Hillegom. Now we have collected a large group of enthusiastic people, and we organize a monthly Arduino café and, of course, the annual World Arduino Day. There is also a lot of interest in the electronics soldering course.”

“There’s not enough DIY electronics being built now, despite the fact that it’s so easy with Arduino.”

Home lab

Ruud has developed a lot of learning material (schematics, PCBs and software) for all of the courses and training sessions. Most of these were created in his home lab (and in the beginning also various things for Siemens). **Figure 1** gives an impression of his home lab.

“I have had my own home lab (or at least a hobby room) since 1965. What you see in the photo is the



Figure 3: The electronics area, with test and measurement equipment.

room I have been using since 2005. The PC part is on the left, the electronics and measurement part is in the middle, and the part for mechanical work is on the right. That’s where I developed my mechanical cleaning robot, for example, as a practical exercise in mechatronics (**Figure 2**).”

“I have developed a lot of test and measurement equipment. Unfortunately, I no longer have my home-made tube oscilloscope, but many designs are still around and have also been built by colleagues. **Figure 3** gives an impression.”

“One of my recent projects is based on the existing DIY calculator from KKmoon. In my opinion, it did not have enough features in its original form, and it was difficult to reprogram. However, it is a nice hardware design, so I developed a new core for it — based on an Arduino board, of course. We are now building a number of these DIY calculators with a group of Arduino students. The calculator functionality (alongside the basic calculator function) is aimed at people who are often involved in programming or other technical activities. **Figure 4** shows the outside and a glimpse of the inside.”

This Arduino Multicalculator will be discussed in more detail in one of the upcoming editions of *Elektor* magazine. ◀

Translated by Kenneth Cox — 230035-01

Questions or Comments

Do you have technical questions or comments regarding this article? Send an e-mail to the editors of *Elektor* at editor@elektor.com.



Figure 2: The cleaning robot.



Figure 4: The Multicalculator: outside and inside.



The “MC

The “MCCAB” (Microcontroller Crash Course for Arduino Beginners) Arduino Nano Training Board was specifically designed by the author and manufactured for Elektor to support the *Microcontrollers Hands-on Course for Arduino Starters*, which comes as a green Elektor guide. The MCCAB board, the guide (in English or German), and an Arduino Nano board are available as a product bundle from the Elektor Store [1].

Entire external modules can also be plugged into the MCCAB via a socket connector, or connected to the microcontroller on the board via the serial interfaces. This eliminates the tedious assembly of experimental circuits, and you can concentrate on the essentials — i.e., your software a.k.a. “the program.” We will thus load the programs created in our exercises into the ATmega328P microcontroller on the MCCAB, where they will be run (i.e., executed as program code).

40 July & August 2023 www.elektormagazine.com

a RESET input for the pushbutton switch on the microcontroller module connected to the MCCAB. (Activation of the latter resets the microcontroller to a defined initial state and restarts the program.) However, the majority of the connections are available as general-purpose inputs/outputs, which the microcontroller can use to connect to the outside world. On the MCCAB, these pins — with just a few exceptions reserved for internal purposes — are brought out to connector strips.

Figure 1 shows a view of the MCCAB with its color-coded function blocks. The operating instructions for the MCCAB, with a detailed description of all components, can be downloaded free of charge from the Elektor website [2].

The MCCAB's Functional Blocks

Referring to Figure 1, the following functional blocks and modules can be distinguished on the MCCAB.

(1) **Arduino Nano microcontroller module** with RESET button (arrow 1a), light-emitting diode L (arrow 1b) and mini USB socket for connection to the user's PC.

(2) Pin headers SV5 and SV6 for **the microcontroller inputs/outputs**. Using Dupont cables, the microcontroller's GPIOs (General Purpose Inputs/Outputs) can be connected to the training board's internal components or to external modules via these two pin headers.

(3) **11 LEDs**, LD10–LD20 (status indicators for the microcontroller's D2–D12 inputs/outputs). Each LED can be connected to the assigned GPIOs, D2–D12, via a jumper on header JP6.

(4) **3×3 LED matrix** LD1–LD9 (nine red LEDs). The column lines are permanently connected to microcontroller outputs D6, D7, and D8, and the row lines can be connected to GPIOs D3, D4, and D5 using the jumpers on socket strip JP1.

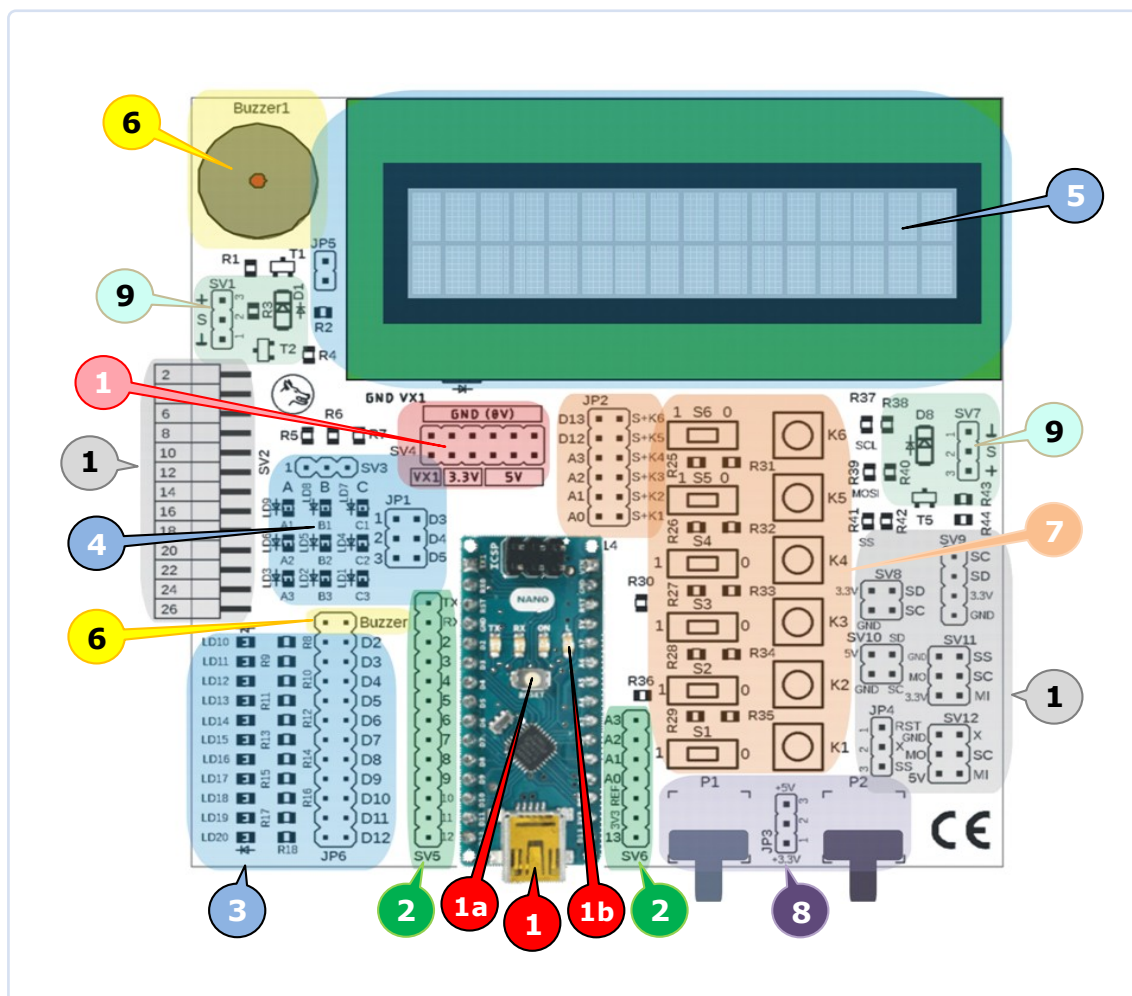
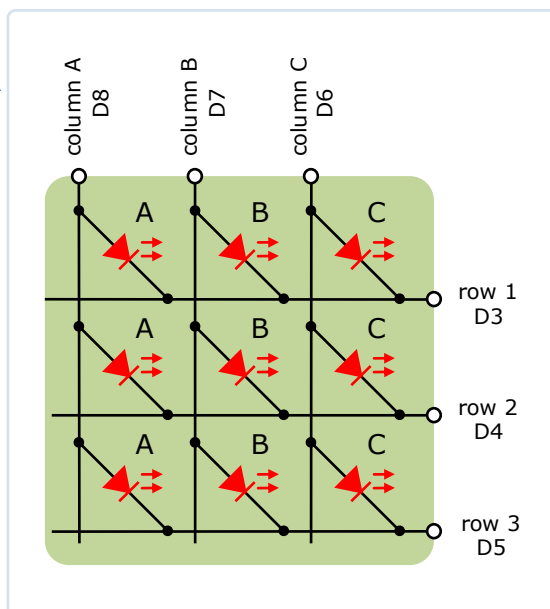


Figure 1: Functional diagram of the "MCCAB" Elektor Arduino Nano Training Board.

Figure 2: 3×3 LED Matrix.



If the 3×3 LED matrix's row lines are either connected to GPIOs D3, D4, and D5 via jumpers on pin header JP1 or to other of the microcontroller's GPIOs via Dupont cables, these row lines, as well as column lines D6, D7, and D8, must never be used for other tasks in a sketch. Double assignment of the matrix GPIOs could lead to malfunctions or even damage to the MCCAB!

(5) **LC Display** with 2×16 characters, connected via the I²C bus to microcontroller pins A4 and A5. By pulling the jumper JP5, the LCD's operating voltage can be switched off for exercises or experiments where the display is not used.

(6) **Piezoelectric buzzer** Buzzer1 can be connected to GPIO D9 via a jumper on the "Buzzer" position of pin header JP6.

(7) **6 slide switches, S1–S6**, connected in parallel to 6 buttons, **K1–K6**. They can be connected to microcontroller inputs A0–A3 and D12–D13 via jumpers on pin header JP2.

(8) **Potentiometers P1 and P2**, whose wipers are connected to the microcontroller's analog input pins, A6 and A7. The 3.3 V or 5 V supply can be applied to the potentiometers via pin header JP3.



Caution: The ATmega328P's A6 and A7 pins are fixed as analog inputs due to the internal chip architecture. Configuring them with the `pinMode()` function is not permitted and can lead to incorrect program behavior.

(9) Pin headers SV1 and SV7 are **switching outputs for external devices**.

(10) Pin headers for serial connection of external **SPI and I²C** modules.

(11) Connector strip SV2 with 2×13 pins for **connecting external modules**.

(12) Pin header SV4 is the **distributor for the board's operating voltages**. The voltages can be connected to internal components on the training board or to external modules using Dupont cables.



The two somewhat more complex functional units, the 3×3 LED matrix (in Figure 1) and the LCD (also in Figure 1) will be explained in more detail below. For all others, please refer to the detailed description in the MCCAB Operating Instructions document [2].



The `MCCAB_Lib` library is available to control the 3×3 LED matrix, LEDs LD10–LD20, buttons K1–K6 and switches S1–S6, as well as Buzzer1. The library can be downloaded free of charge by MCCAB owners and integrated into the Arduino IDE.

To control the LCD, we use the `LiquidCrystal_I2C` library, which can be downloaded free of charge from the internet and "inserted" into the Arduino IDE.

The 3×3 LED Matrix

The MCCAB contains nine light-emitting diodes positioned in a matrix (in Figure 1). **Figure 2** shows their basic circuitry.

The matrix consists of three columns and three rows. The columns are labeled A, B, and C, while the rows are numbered 1, 2, and 3. At each of the nine column/row intersections, a light-emitting diode is connected — its anode to the column line and its cathode to the row line. The nine LEDs are labeled according to their respective column/row position, e.g., LED "B2" is connected to Column B and Row 2. For an LED to light up, its column line must be at logic 1 (+5 V) and its row line at logic 0 (0 V).

The instruction manual for the MCCAB states that the column leads are permanently connected to microcontroller GPIOs D6–D8. The row leads can be connected to GPIOs D3–D5 using jumpers on header JP1.

If the 3×3 LED matrix's row lines are either connected to GPIOs D3, D4, and D5 via jumpers on pin header JP1, or to the microcontroller's other GPIOs using Dupont cables, the row lines and column lines D6–D8 must not be used for other tasks in a sketch. Connecting the matrix GPIOs in this way would lead to malfunctions or, in the worst case, even damage to the MCCAB! **If the matrix is not used in a sketch, jumpers on MCCAB header JP1 should be removed.**

Benefits of the matrix arrangement

We are familiar with LED matrices from perimeter advertising in football stadiums, for example, where colored high-power LEDs are used in a matrix arrangement to generate apparently moving images.

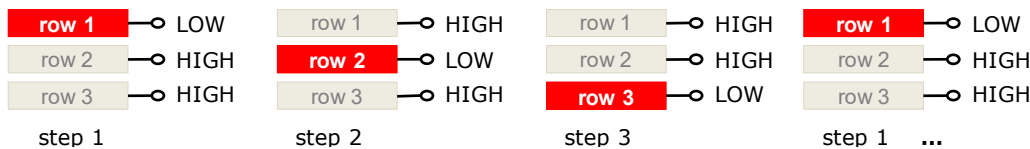


Figure 3: Periodic stepwise activation of the three rows that make up the matrix.

If we were to control nine LEDs individually, nine of the microcontroller's GPIOs would be required. By arranging the LEDs in a matrix, we only need six. The more LEDs to be controlled, the greater the benefit of matrix control gets: With a matrix consisting of eight columns and eight rows, 64 LEDs can be controlled, whereas with individual control lines, 48 additional GPIOs would be required!

Controlling the matrix in multiplex mode

Figure 2 shows that each of the matrix's column lines and row lines on the MCCAB is connected to three LEDs. Consequently, simultaneous control of all rows and columns would not work, as LEDs that should actually be switched off would also light up unintentionally. Instead, as shown in **Figure 3**, only one row line may be active at a time and the column lines must apply the currently activated row's LED bit pattern. The other two row lines must be open during this time or switched off with a logic High level so that no current can flow through them.

If the three lines are activated cyclically in a sufficiently rapid sequence as in Figure 3, due to the perceptual inertia of the human eye, you see what appears to be a static image of all nine LEDs. The user program controls the LED matrix in an endless loop, in which one of the three rows, 1, 2, or 3, is set to logic Low potential cyclically while the other two rows are set to High level. The column connections of all LEDs that should light up in the currently active row are set to High level. The LEDs that need to be off in the active row need their column connections set to logic Low.

For example, to light up two LEDs, A3 and C3, Row 3 must be at Low level and Columns A and C at High, while the two row lines, 1 and 2, are High and Column B is Low.

The LC (Liquid Crystal) Display

The Elektor MCCAB is equipped with an LC Display (LCD) that enables output of text, numerical values, and even self-defined special characters. The display used has two lines of 16 columns. One character can be displayed in each column. Each character is formed from the points of a 5×8 matrix, as shown in **Figure 4**. The required 5×8 dot matrix bit patterns for each individual character are stored inside the LCD controller according to the ASCII table.

Now, ASCII is a 7-bit code, while our ATmega328P microcontroller is an "8-bit machine," meaning that it can process a byte in one step and also stores its data in byte format. Thus, the eighth bit would typically be set to 0 when storing ASCII codes. However, the developers of the HD44780 display controller (which is standard in this category and can be found on almost all LCD modules) did not want to leave this eighth bit of the byte unused, and extended the ASCII character set by another 128 character codes (where the eighth bit is at logic 1). For this reason, our LCD also uses the character codes 128 to 255 (more or less contiguously).

In addition to the 95 printable characters with the codes 32–126, ASCII also has 33 control characters, i.e. 0–31 and 127, which are "not printable." Because these characters are intended for control purposes, the display does not display them. The HD44780 display controller is available in two versions: Either with the A00 ROM code or with the A02 ROM code. Version A00 displays blank characters for codes 106–31 as well as 128–159. Version A02, on the other hand, displays special characters for these address spaces (see **Table 1**). Which of the two ROM versions is installed in the MCCAB's LCD cannot be predicted, as the manufacturer of the LCD module makes this selection.

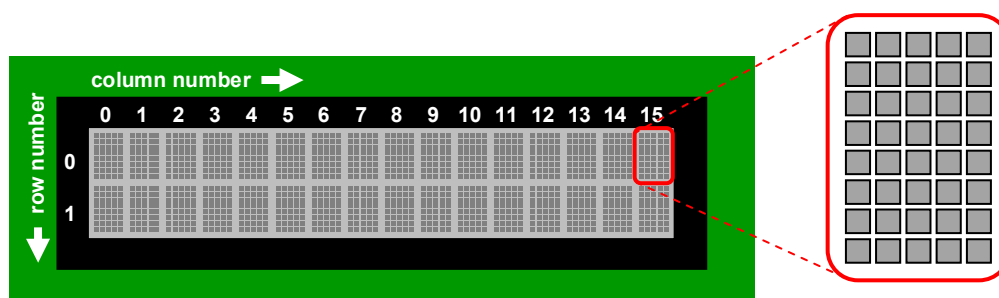


Figure 4: Illustration of the LC Display on the MCCAB with its 16×2 displayable characters.

Table 1: Character memory allocation for the two HD44780 variants.

Character Code	ROM Code A00	ROM Code A02
0–7	user-defined characters	user-defined characters
8–15	user-defined characters	user-defined characters
16–31	(no display)	visible characters
32–127	visible characters	visible characters
128–159	(no display)	visible characters
160–255	visible characters	visible characters

The user can define up to eight special characters and display them on the LCD. ASCII codes 0 through 7 are assigned to these eight special characters. Optionally, the same eight special characters can also be addressed via ASCII codes 8 through 15.

Figure 4 also shows the numbering of the rows and columns of the display, which, in both cases, starts with 0. By specifying these coordinates, a character can be written specifically to a certain position on the display. For this purpose, the LCD has a cursor that determines the position of the character to be written. The LCD library described in the book contains methods for positioning this cursor.

Although the display can only show 16 characters in each row, the memory in which the characters are stored in the display controller in fact has 40 memory locations for each line. As you can see in **Figure 5**, there is a gap of 24 memory locations between the last display address, 39, of the first line and the start address, 64, of the second line.

By means of *shift* functions, which are contained in the LCD library, the memory area visible in the display can be shifted over the entire memory area like a window.

The image in Figure 5 has three levels:

- at the top: the visible window of the display in the basic setting without previous slide operations.
- in the middle: the display content of the basic setting after a slide operation to the left.
- at the bottom: the display content of the basic setting after a slide operation to the right.

Setting the LCD contrast

The MCCAB is supplied with no contrast set for the LCD. For this reason, and because the contrast of the display can drift due to environmental conditions (such as temperature) or aging, the LCD has a trim potentiometer for you to adjust the contrast. The little pot is accessible from the underside of the Training Board and marked with an arrow. Adjust the contrast with a small screwdriver during the first start-up or if otherwise required.

i Hint: If no characters are visible on the display after text output when using the LCD for the first time, it is most likely due to a lack of LCD contrast adjustment!

Data transmission from the microcontroller to the LCD

On each LCD module, there is a HD44780 display controller, which receives the (ASCII) codes of the characters sent by the microcontroller via an interface, generates the corresponding characters of the

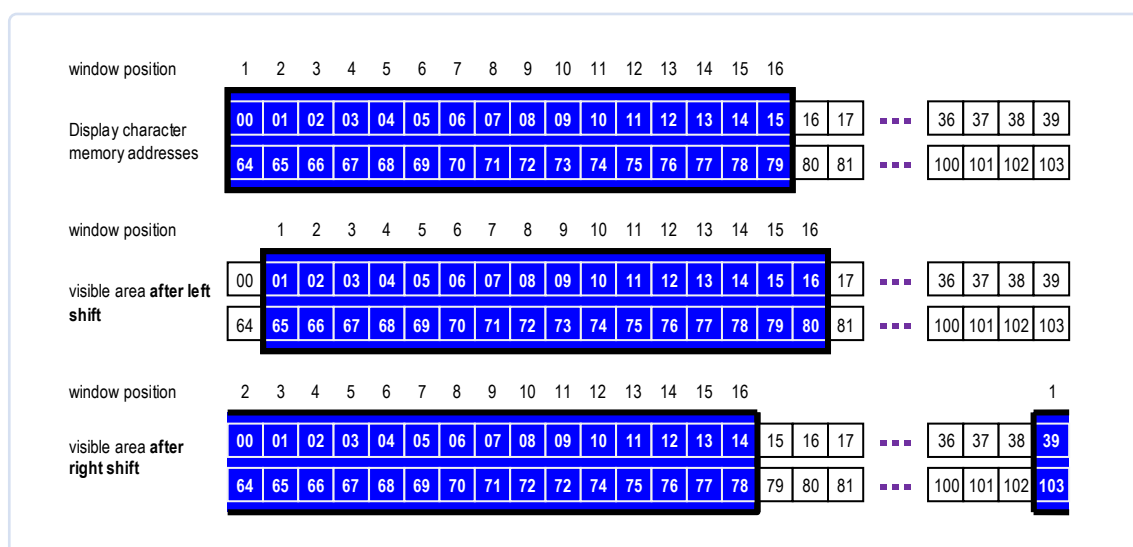


Figure 5: The display's visible content before and after shifting.

5×8 dot matrix mentioned above from the codes, and displays them.

The LCD controller type HD44780 only has a parallel interface for receiving the data to be displayed, i.e., the microcontroller writes a data byte consisting of eight bits, D0–D7, into the HD44780 display controller's input register and then onto the LCD using three control signals: RS, RW, and E, as shown in **Figure 6**. This method of parallel data transfer would considerably limit the resources on the MCCAB because the LCD alone would already occupy 11 of the ATmega328P microcontroller's 16 freely available pins!

The second option for data transmission to the HD44780 controller, in which the eight data bits are transmitted in two packets of four bits each, does not really help either because, even in this case, seven pins of the microcontroller would still be used by the LCD, since the three control signals, RS, RW, E are also required, in this case.

For this reason, the LCD module on the MCCAB is controlled via the I²C bus, a synchronous serial bus that consists of only one data line (SDA), one clock line (SCL), and it transmits data bit by bit. The data traffic via the I²C interface is carried out using two lines from the ATmega328P microcontroller, A4 (SDA) and A5 (SCL), as shown in **Figure 7**.

An additional adapter on the bottom of the LCD module converts the I²C signals into parallel signals. This adapter enables the abovementioned method of transmitting two sets of four bits in succession via the data lines, D4–D7, i.e., first data bits D4–D7 and then data bits D0–D3 are transmitted over the same lines, D4–D7.

Since the microcontroller's A4 and A5 lines on the MCCAB are reserved for the I²C interface anyway, no resources are lost with this type of data transmission to the LCD because, in principle, several participants can be connected to the I²C bus at the same time. Since each participant connected to the bus resides at its own I²C address, it only "senses" that it is being addressed when a data packet arrives with the address belonging to this participant. The LCD

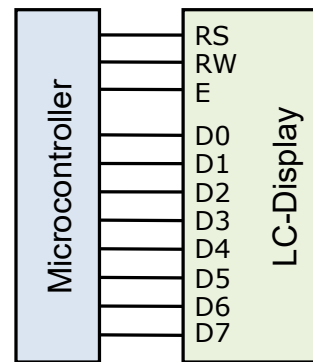


Figure 6: LCD control with eight data bits and three control lines.

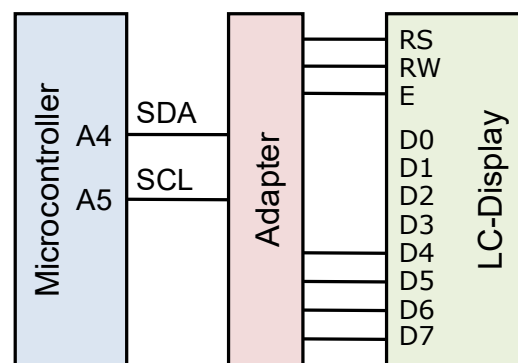


Figure 7: LCD control via the I²C bus rather than in parallel (cf. Figure 6).

on the MCCAB usually resides at I²C address 0x27. If the address differs due to the manufacturer, this is indicated on the display. ◀

230215-01

Questions or Comments?

If you have technical questions about this article, feel free to contact Elektor on email editor@elektor.com.



Related Products

- **Microcontrollers Hands-on Course for Arduino Starters (Bundle) (SKU 20440)**
 - 1 × Guide: *Microcontrollers Hands-On Course for Arduino Starters*
 - 1 × MCCAB Arduino Training Board
 - 1 × Arduino Nano<https://elektor.com/20440>

WEB LINKS

- [1] Elektor Arduino Nano Training Board MCCAB and Companion Guide: <https://elektor.com/20440>
- [2] Instruction Manual for Elektor Arduino Nano Training Board MCCAB: <https://elektor.com/20440>



Source: Shutterstock / Metamorphosis

From Life's Experience

Modern Luddism

By Ilse Joostens (Belgium)

It's true that hardware performance is constantly increasing, which also allows the development of ever-smarter software applications. When I was working as a system administrator around 20 years ago, I was crazy about new technology and my personal mission was to make ordinary computer users enthusiastic about new technology. That was not always so easy, to put it mildly. Now that I am a bit older, and, in particular, wiser, my enthusiasm for new technology has cooled down a bit, and I have an uneasy feeling about recent developments in AI.

Artificial Stupidity

Unless you have been living in a cave in recent months (without Internet), you will have certainly heard about the new technological (r)evolution in artificial intelligence, and, more specifically, OpenAI's chatbot, ChatGPT [1]. There is enormous interest, and Microsoft is fully committed to generative AI with a \$10 million investment in this startup. In time-honored tradition, their rival, Google, doesn't

want to be left behind and is presently working hard on a similar bot with the melodious name of Bard. Even Baidu is joining the fray and has started developing a Chinese version of ChatGPT called Ernie.

As an amateur photographer, I was already interested in DALL-E [2], StabilityAI [3] and Fotor. These applications can convert text into images and modify — or even combine —

existing images. It became a bit of an addiction, and I spent hours experimenting with them, until smoke came out of my keyboard (**Figure 1**). The results ranged from disappointing to stunning, and, sometimes, even frightening. As regards the latter, my preference for the horror genre may offer an underlying explanation. This technology is also at the heart of the creepypasta around "Loab" [4][5], a demonic woman said to be hiding deep in the caverns of AI.

Naturally, out of curiosity, I also tried out ChatGPT, and, at first glance, the bot reminded me of "Evarist the computer," a character from the Flemish youth series, *Merlina* [6], whom you could ask anything. The answers to my questions were okay linguistically, but, unlike Evarist, ChatGPT often missed the mark and some of the answers were monumentally hilarious — in the style of Xavier De Baere (*a well-known Flemish TV comedian – ed.*). Although not the original intention, it looks like the bot can also generate source code. A simple Arduino project was not a problem (**Figure 2**), but a more complex request to write a C program for an AVR to initialize a WIZnet W5500 Ethernet IC with a fixed IP address, without using any libraries, got stuck time after



Figure 1: AI creation by Ilse Joostens.

time. Maybe the paid version with OpenAI's GPT-4 would do better, but that is not an option for an application that I do not (yet) have any specific use for. I can get the device working myself, and the more mundane questions of everyday life, such as how to whip up a three-course vegetarian dinner with a chocolate dessert, or handiwork ideas for toddlers, are not something I am especially interested in.

Just as "cloud" is a buzzword for working and storing data via the internet, and the catchword "IoT" means devices, such as "smart" lamps, connected to the internet, "artificial intelligence" is a blanket term for powerful algorithms and models linked to big data. In the case of GPT-3, this involves 570 gigabytes of input, while DALL-E 2 makes do with more than 650 million images. The term "intelligence" is not really relevant, and researchers who train neural networks to look for connections that don't really exist are actually creating "artificial stupidity," if you will [7].

The Dark Side of AI

I admit that generative AI inspires and gives new insights, but I'm not such a great fan,

considering that the generated texts and images are essentially plagiarism because they are based on the work of millions of people. Another thing is that any AI that is trained based on historical data, which inevitably contains human biases, takes over these biases — sometimes in amplified form. I don't think that ChatGPT will be writing this column in the future, but things are already pretty difficult for online sales of photographs and works of art. Along with constantly improving cameras in smartphones, this is a new setback for amateur photographers and artists looking to make a bit of extra money [8].

There are also many opportunities for misuse, ranging from letter-perfect phishing emails to hate speech and deep fakes. Thanks to ChatGPT, AI is now a hot item, but it has been around for a lot longer and can also be used — or misused — for other purposes, for example by government agencies such as in China with their social credit system or in the US where

facial recognition technology is being used to identify demonstrators. In Flanders, too, AI is being applied to aerial photos to see whether someone has cut down more trees in their garden than allowed. Personally, I have been the victim of crime several times in the past, but there was never any effort put into finding the perpetrators. By contrast, all means are good when it comes to ascertaining minor offenses committed by good citizens, with, as a sad high point, the case of a man from Heusden-Zolder who, on the way home from a restaurant with his family, stopped briefly to toss a full diaper from his sick daughter into a public waste bin. To his surprise, a while later he received a twenty-page (!) police report in the mail, including photos, for illegal dumping [9]. Apparently, locking up real criminals does not put enough money in the till.

It's more than enough to turn you into a modern Luddite. ◀

230196-01

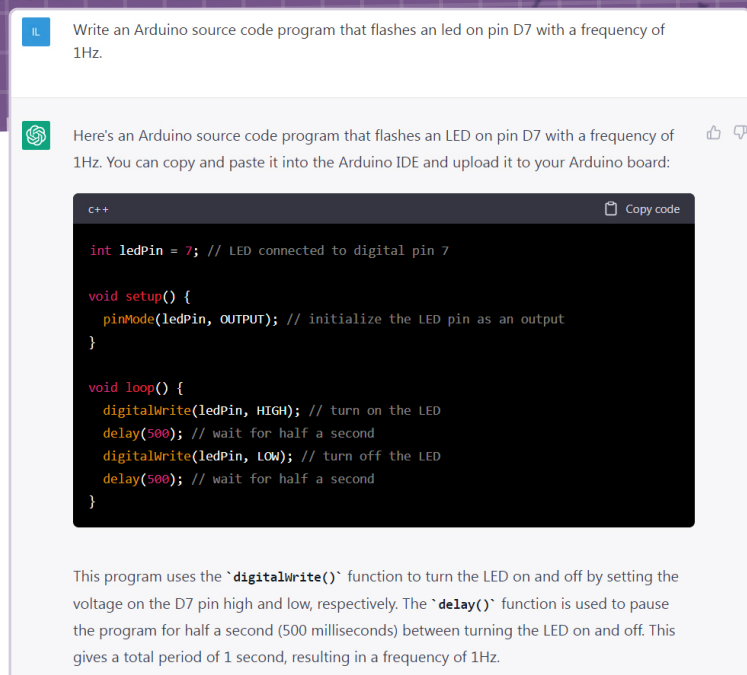
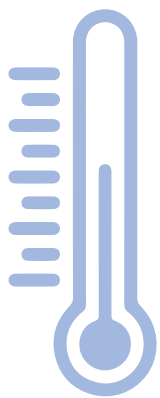


Figure 2: ChatGPT writes software...

WEB LINKS

- [1] ChatGPT: <https://chat.openai.com>
- [2] DALL-E 2: <https://labs.openai.com>
- [3] StabilityAI's Stable Diffusion: <https://huggingface.co/spaces/stabilityai/stable-diffusion>
- [4] Loab, an AI-generated entity: <https://loab.ai>
- [5] YouTube: The Disturbing Art of AI (spooky!): <https://youtu.be/i9InAbpM7mU>
- [6] Merlina: Remembering the hit BRT youth series from the 1980s: <https://merlina.info/evarest/uitvindingen>
- [7] EOS Wetenschap: Kunstmatige Domheid [Artificial Stupidity, Dutch]: <https://eoswetenschap.eu/technologie/kunstmatige-domheid>
- [8] YouTube: AI Art Apocalypse: <https://youtu.be/HDbu2rvhCk4>
- [9] Nieuwsblad: Heusdens gezin riskeert GAS-boete voor weggooien van pamber [Dutch]: https://nieuwsblad.be/cnt/dmf20220901_96655325



Sensor 101:

The DS18B20

Temperature Sensor

Connection to the 1-Wire Bus

By Mathias Claussen (Germany)

The Dallas Semiconductor DS18B20 sensor is included in many beginner kits and offers an easy introduction to temperature measurement. Before you dive in, it's helpful to have some knowledge about the 1-Wire bus and how to connect the sensor. In this article, we'll take a quick excursion into the basics of using the DS18B20, so you can start measuring temperature with confidence!

If you're looking to record temperatures using a microcontroller, you have a variety of sensors and bus systems to choose from. One popular choice is the Dallas Semiconductor DS18B20. It has a range from -5°C to 125°C (-67°F to $+275^{\circ}\text{F}$) with an impressive accuracy of $\pm 0.5^{\circ}\text{C}$. Its versatility makes it suitable for measuring not only ambient temperatures, but also for monitoring freezers and cold rooms. In this article, we take a closer look at how it works with a microcontroller, along with examples featuring source code and circuit diagrams to show how you can integrate it into your own projects.

The DS18B20

The DS18B20 sensor utilizes the 1-Wire bus system, patented by Dallas Semiconductor in 1989, and allows for the connection of multiple sensors. With a power supply range of 3.0 V to 5.5 V, the DS18B20 can be connected directly to GPIOs on a range of devices, from the Arduino UNO to the Raspberry Pi Pico. The 1-Wire bus also offers a parasite supply mode, where energy is drawn from the data line to power the sensor. This means that the 1-Wire bus only requires one pin on a microcontroller (MCU) to connect many sensors, making it a popular choice, especially on smaller MCUs boards with fewer GPIOs. While other 1-Wire sensors exist, we will just concentrate on the Dallas device here.

The DS18B20 sensor is available in different forms, as demonstrated in **Figures 1 and 2**. Thanks to the 1-Wire bus, it is now very easy



Figure 1: The DS18B20 in a TO-92 package.



Figure 2: Waterproof version of the DS18B20 with cable.

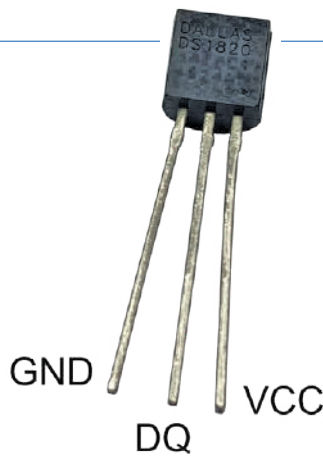


Figure 3: DS18B20 pinout.

to connect a larger number of sensors. Although the 1-Wire protocol itself does not impose a limit on the number of sensors that can be connected to the bus, limits are determined by the bus's electrical properties.

The 1-Wire Bus

To connect sensors to the 1-Wire bus, three wires are required: Ground (GND), data (DQ), and supply voltage (VCC). The 1-Wire bus is bidirectional and operates on a controller/target concept (master and slave), where controller and target exchange data via the data line. The DS18B20 sensor pinout can be found in **Figure 3**.

All nodes on the single data line use an open-collector driver; in the event of a data collision, where two nodes mistakenly try to

talk at the same time, the result is just corrupted data. There can never be a hardware conflict where one node is trying to pull the data line high while another is trying to pull it low. A pull-up resistor is needed because none of the nodes can actively pull the bus to VCC. **Figure 4** shows the connection of a 1-Wire sensor to an Arduino UNO.

In addition to the electrical connection, a protocol is required for the operation of 1-Wire devices. Fortunately, most microcontrollers include a UART that can be used for this purpose, without the need for any special hardware. A suitable circuit can be seen in **Figure 5**.

For control via UART, Analog Devices provides a useful article [1]. While the UART can help relieve CPU loading, it is not essential with today's MCUs. Even a small ATtiny has the resources to address sensors connected via the 1-Wire bus using a purely software, bit-banging solution, requiring only one I/O pin. Most libraries that work with the DS18B20 just use a single GPIO, making the use of a UART something of a rarity.

Sensor Identification

When more than one sensor is connected to a common bus wire, each sensor must be individually addressable so that only one sensor is allowed to write to the bus at a time and its data cannot be confused or corrupted by another sensor. For this purpose, each participant on the 1-Wire bus has a unique, 64-bit-wide identification code (UUID). This consists of an 8-bit code indicating

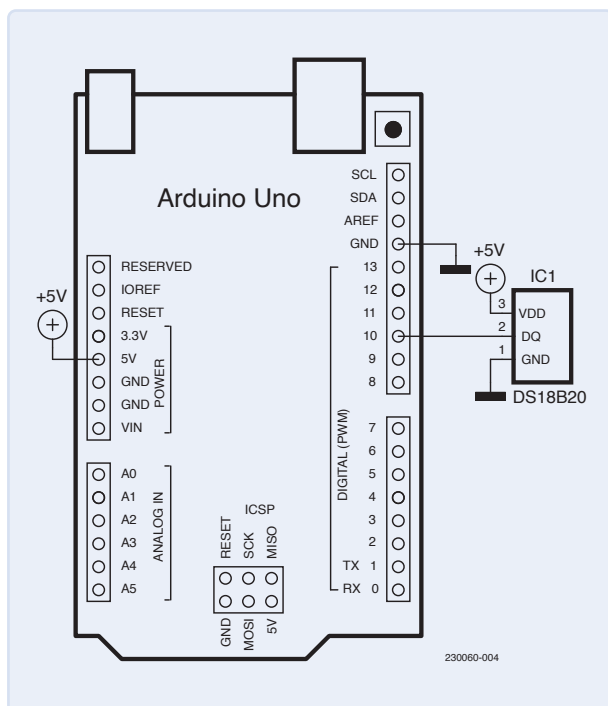


Figure 4: Circuit diagram of Arduino UNO with a DS18B20.

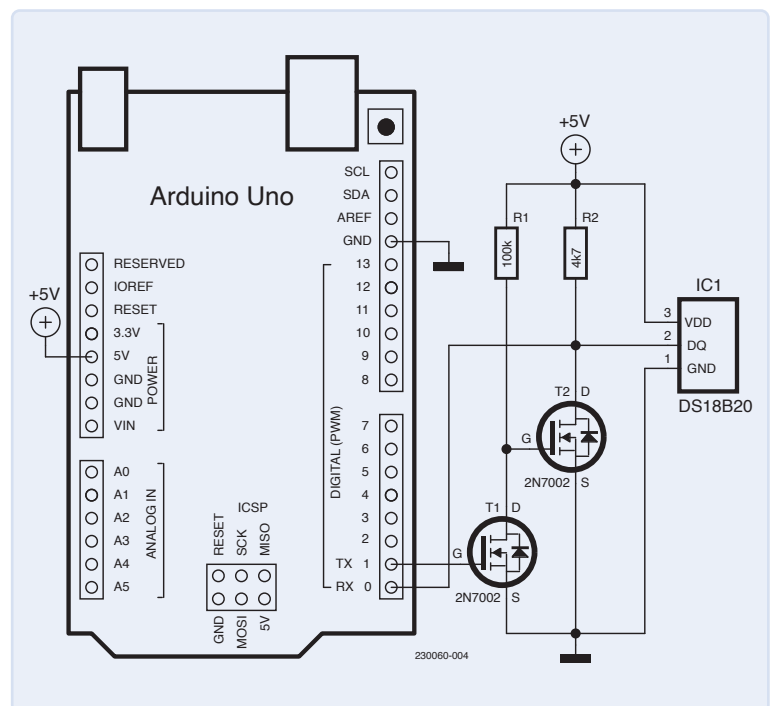


Figure 5: Schematic of DS18B20 1-Wire bus using the UART.

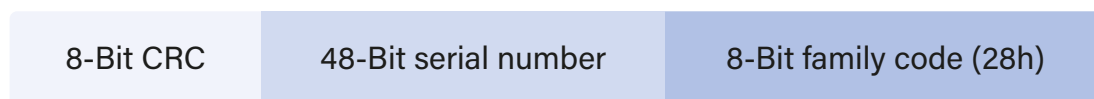


Figure 6: Format of the DS18B20 UUID message.

the sensor family, a 48-bit serial number and an 8-bit checksum (CRC) (**Figure 6**). The sensor serial number, however, is not printed anywhere on the sensor but resides internally, stored as 8 bytes in the ROM. It can be determined by software using a search function on the 1-Wire bus. The search algorithm can be found on the Analog Devices page [2].

Example Code for Arduino

Based on the circuit diagram in Figure 4, the setup is kept very simple. The sensor uses a 5 V supply like the Arduino UNO. The Arduino UNO GPIO connected to the data or DQ line requires a 4.7 kΩ pull-up resistor. The 1-Wire bus library we use in this case is the *OneWire* library by Paul Stoffregen [3]. The library's sample code also includes reading data from the DS18B20 sensors (**Listing 1**), which we can now look at in a little more detail.

To use the library, it first needs to be included using `#include <OneWire.h>` at the beginning of the sketch. Next, with `OneWire ds(10)`, the 1-Wire bus data line is assigned for operation with pin 10. With `ds.search(addr)` the next sensor on the bus is sought. When all the sensors have been identified, the search is terminated. The identification of any sensor found is stored in `addr`.

DS18B20 - The Clone Wars

The DS18B20 is a popular sensor that is often included in various sensor kits and microcontroller starter kits. These sensors are also really cheap; you can pick one up for as little 30 cents on certain online marketplaces. Distributors such as Mouser or Farnell, however, will charge you well over 4 euro for a single DS18B20.

The popularity of this sensor has made it worthwhile for some rogue chip fab plants to come up with their own versions, which look like and seem to behave like the original sensor, but may deviate significantly from the original's characteristics upon closer inspection. Recognized distributors source their components from certified suppliers.

To ensure that you have an authentic sensor, Chris Petrich's GitHub page [4] provides Arduino sketches for testing your sensor, as well as more information about individual clones and their deviations from the original design.

Now that the sensor has been identified, it can be interacted with. The first byte, `addr[0]`, contains the code indicating the family. This can be used to identify whether a DS18B20, DS18S20, or DS1822 temperature sensor has been found, or whether it is a different type of 1-Wire bus device.

The following sequence can be seen in line 69:

```
ds.reset();
ds.select(addr);
ds.write(0x44, 1);
```

A reset of all bus participants is carried out once using `ds.reset()`. Then, afterwards using `ds.select(addr)`, the discovered sensor is addressed. To request a temperature reading from the sensor, it is necessary to initiate a measurement first. The corresponding command for this is `ds.write(0x44, 1)`, where 0x44 is the command recognized in the sensor. With the second parameter (here, 1) the I/O pin is actively driven high to supply sensors like the DS18S20 with parasite power. The result of the temperature measurement is available after 750 to 1000 ms. In this example, a delay is used to wait for the processing to complete.

Once the temperature measurement is complete and the result is ready, it can be fetched from the sensor. The sensor is addressed again using `ds.select(addr)` and instructed that now the value stored in its scratchpad memory is to be read out (`ds.write(0xBE)`). The nine bytes stored here, including the temperature reading, will now be sent out.

With the DS18B20, the temperature is available after readout as a 16-bit value, made up of two register bytes. If a DS18S20 type sensor is used instead of a DS18B20, the register values are weighted differently so that their position in the register needs to be modified. The code example takes this into account by performing the necessary 3 left-shifts on the raw value.

The DS18B20 outputs the temperature with 12-bit resolution and takes 750 ms for the conversion. It can optionally output 11, 10, or 9-bit values, to give a correspondingly faster conversion time. At 9-bit resolution, the value of the low bits are undefined, so they can be zeroed.

The sensor temperature reading is always in Celsius; your software will need to do the conversion to Fahrenheit if this is required.

Summary

Connecting a temperature sensor to a microcontroller is easy using the DS18B20 with 1-Wire. As we saw, using the Arduino platform, there are a whole bunch of libraries and source code examples to



streamline the process. With this setup, only one pin of the MCU is needed to communicate with the sensor, and multiple sensors can be connected, allowing a small MCU with one spare GPIO to support several sensors simultaneously. ◀

Translated by Martin Cooke — 230060-01

Questions or Comments?

If you have any technical questions or comments about this article please contact the Elektor editorial team at editor@elektor.com.



Related Products

- > **Elektor 37-in-1 Sensor Kit (SKU 16843)**
<https://elektor.com/16843>
- > **Cytron Maker UNO (SKU 18634)**
<https://elektor.com/18634>
- > **MakePython ESP32 Development Kit (SKU 20137)**
<https://elektor.com/20137>

WEB LINKS

- [1] Using a UART to implement a 1-wire-bus-master, Analog Devices:
<https://analog.com/en/technical-articles/using-a-uart-to-implement-a-1wire-bus-master.html>
- [2] 1-Wire search algorithm, Analog Devices: <https://analog.com/en/app-notes/1wire-search-algorithm.html>
- [3] OneWireLibrary by Paul Stoffregen on GitHub: <https://github.com/PaulStoffregen/OneWire>
- [4] Chris Petrich, "Your DS18B20 temperature sensor is likely a fake, counterfeit, clone...", GitHub:
https://github.com/cpetrich/counterfeit_DS18B20



Listing 1: DS18B20 Arduino example.

```
001      #include <OneWire.h>
002
003      // OneWire DS18S20, DS18B20, DS1822 Temperature Example
004      //
005      // http://www.pjrc.com/teensy/td_libs_OneWire.html
006      //
007      // The DallasTemperature library can do all this work for you!
008      // https://github.com/milesburton/Arduino-Temperature-Control-Library
009
010      OneWire ds(10); // on pin 10 (a 4k7 resistor is necessary)
011
012      void setup(void) {
013          Serial.begin(9600);
014      }
015
016      void loop(void) {
017
018          byte i;
019          byte present = 0;
020          byte type_s;
021          byte data[9];
022          byte addr[8];
023          float celsius, fahrenheit;
024
025          if (!ds.search(addr)) {
026              Serial.println("No more addresses.");
027              Serial.println();
```

continued overleaf

```

028         ds.reset_search();
029         delay(250);
030         return;
031     }
032
033
034     Serial.print("ROM =");
035     for(i = 0; i < 8; i++) {
036         Serial.write(' ');
037         Serial.print(addr[i], HEX);
038     }
039
040     if (OneWire::crc8(addr, 7) != addr[7]) {
041         Serial.println("CRC is not valid!");
042         return;
043     }
044
045     Serial.println();
046     // the first ROM byte indicates which chip
047     switch (addr[0]) {
048
049         case 0x10:
050             Serial.println("  Chip = DS18S20"); // or old DS1820
051             type_s = 1;
052             break;
053
054         case 0x28:
055             Serial.println("  Chip = DS18B20");
056             type_s = 0;
057             break;
058
059         case 0x22:
060             Serial.println("  Chip = DS1822");
061             type_s = 0;
062             break;
063
064         default:
065             Serial.println("Device is not a DS18x20 family device.");
066             return;
067     }
068
069     ds.reset();
070     ds.select(addr);
071     ds.write(0x44, 1); // start conversion, with parasite power on at the end
072     delay(1000);       // maybe 750ms is enough, maybe not
073     // we might do a ds.depower() here, but the reset will take care of it.
074
075     present = ds.reset();
076     ds.select(addr);
077     ds.write(0xBE);     // Read Scratchpad
078
079     Serial.print("  Data = ");
080     Serial.print(present, HEX);
081     Serial.print(" ");
082     for (i = 0; i < 9; i++) {           // we need 9 bytes
083         data[i] = ds.read();
084         Serial.print(data[i], HEX);
085         Serial.print(" ");
086     }
087     Serial.print(" CRC=");
088     Serial.print(OneWire::crc8(data, 8), HEX);

```




```
089     Serial.println();
090     // Convert the data to actual temperature
091     // because the result is a 16-bit signed integer, it should
092     // be stored to an "int16_t" type, which is always 16 bits,
093     // even when compiled on a 32-bit processor.
094
095     int16_t raw = (data[1] << 8) | data[0];
096     if (type_s) {
097         raw = raw << 3; // 9 bit resolution default
098         if (data[7] == 0x10) {
099             // "count remain" gives full 12 bit resolution
100             raw = (raw & 0xFFF0) + 12 - data[6];
101         }
102     } else {
103         byte cfg = (data[4] & 0x60);
104         // at lower res, the low bits are undefined, so let's zero them
105         if (cfg == 0x00) raw = raw & ~7; // 9 bit resolution, 93.75 ms
106         else if (cfg == 0x20) raw = raw & ~3; // 10 bit res, 187.5 ms
107         else if (cfg == 0x40) raw = raw & ~1; // 11 bit res, 375 ms
108         // default is 12-bit resolution, 750 ms conversion time
109     }
110
111     celsius = (float)raw / 16.0;
112     fahrenheit = celsius * 1.8 + 32.0;
113     Serial.print(" Temperature = ");
114     Serial.print(celsius);
115     Serial.print(" Celsius, ");
116     Serial.print(fahrenheit);
117     Serial.println(" Fahrenheit");
118 }
```

YOUR KEY TO CELLULAR TECHNOLOGY



WÜRTH
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

Is Matter the Thread to Save the Smart Home?

New Standards to Simplify the Smart Home

By Stuart Cording (Elektor)

Swipe, press, select, back, select, swipe – ah, the lights are now on. Well, most of them. The Smart Home isn't always as smart as promised, thanks to too many apps, compatibility issues, and standards. Well, now there is a new solution – some more standards! But will Thread and Matter make a difference?

After a hard day's work, what could be better than coming home to a house that's ready for you? The sound system has your favorite music on, the rooms are warm and comfortable, and your meal is ready to take out of the oven. As you enter the living room, the blinds close to block out the direct sunlight, and the standing lamp by your comfy armchair sparkles into life so that you can continue reading your novel on your e-reader. Naturally, this has also sprung to life, displaying the page where you left off last night.

Ahh, the dream of the Smart Home. And, for many, a dream it remains. While there are wireless and wired protocols aplenty, each has its own domain, apps, and features. This vision of homes responding to our needs has a long history. In the late 1990s, Microsoft shared a video demonstrating, pretty accurately, much of the

technology we have today. In a promotion video from the era, they show electronic door locks, internet shopping, voice assistants, and much more, all powered by Windows XP and CE [1]. And these ideas weren't new, either. Ray Bradbury, the celebrated 20th-century science fiction writer, conjured up an almost identical home in his 1950s short story, "There Will Come Soft Rains." [2]. However, before we learn how Thread and Matter are supposed to resolve today's complexity, we need to discover how we got here.

Nascent Short-Range Wireless

As the Internet of Things became, well, a thing, it was clear that those things would need to be connected wirelessly. Back in the early 2000s, there weren't many choices. You had Wi-Fi, a GSM cellular modem, or Bluetooth. The first two weren't known for low power, so they were essentially out for battery-power devices. That just left Bluetooth, a technology that aligned well with the needs of the nascent mobile phone market and a promising alternative to IrDA, the infrared link used with personal digital assistants (PDAs) for data syncing.

Relying on the 100 MHz bandwidth around 2.4 GHz available worldwide as part of the industrial, scientific, and medical (ISM) band, Bluetooth looked poised to own IoT. At the time, there were Internet gateways, wireless serial links, and even Bluetooth attachments for printers. It wasn't even supported natively by Windows, requiring software supplied with a dongle to enable communication between a PC and a phone.

Bluetooth excelled in audio, providing a robust interface for hands-free telephony with single-ear headsets and vehicle accessories. Car phones and bulky plastic adapters became a thing of the past as drivers simply paired the phone they owned with the vehicle's hands-free kit.

The protocol was defined as a personal area network (PAN) from its inception. The specification allows up to seven slave devices to connect to a master to form a piconet (**Figure 1**). There was even provision for a scatternet, formed by a master of one piconet being a slave of another or a slave being part of two piconets (**Figure 2**). However, with only 720 kb/s of throughput available at the time, the data rate dropped off significantly as the network grew and master/slave role switching was taken into consideration. Furthermore, there was no definition for how data would pass between piconets — that was left to the implementer. The author was involved in several projects during this early time, but scatternets never seemed possible in practice.

Saving Power

Bluetooth's need to keep the transceiver powered made it unsuited to battery-powered sensors. Nokia had been developing a 2.4 GHz technology called Wibree [3] to address this. By the mid-2000s, dual-mode chipsets offered Bluetooth and Wibree side by side, targeting smart watches, sport sensors, and even wireless keyboards. In 2007, Wibree was adopted by those maintaining the Bluetooth specification,

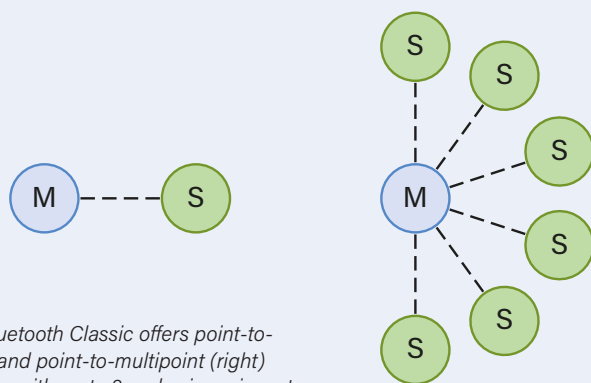


Figure 1: Bluetooth Classic offers point-to-point (left) and point-to-multipoint (right) connectivity, with up to 8 nodes in a piconet.

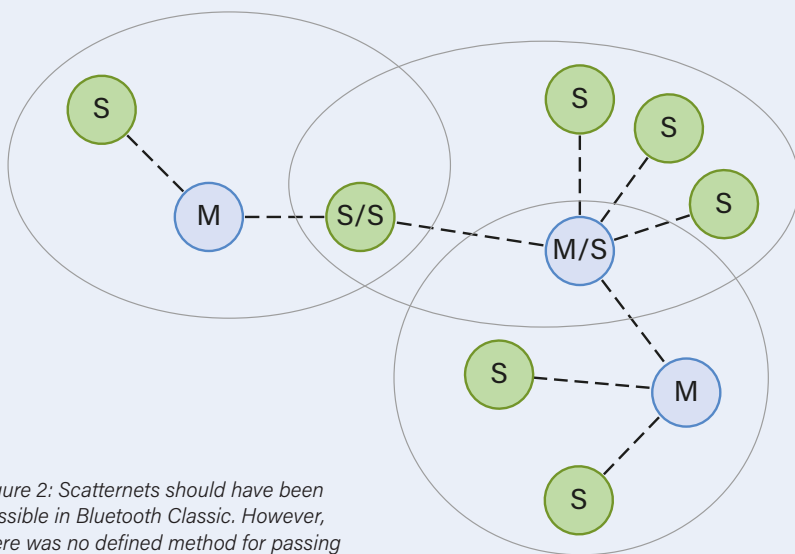


Figure 2: Scatternets should have been possible in Bluetooth Classic. However, there was no defined method for passing data between piconets.

had been around since 2003, and delivered a revised specification in 2006 (**Figure 3**). This also operated in the 2.4 GHz band, but other frequencies, such as 784 MHz (China), 868 MHz (Europe), and 915 MHz (Australia and the US), could also be used. There was also Z-Wave, a sub-gigahertz technology developed by Zensys that was eventually bought by Silicon Labs, which also launched around 2003.

Both technologies offer self-healing mesh networks, meaning that even if a node fails, network traffic will find an alternative path between the device sending the message and the device that should receive it. This happens in the background and doesn't involve the system user. Today, both these wireless systems have several thousand products using them.

Problems with Low-Power Mesh Networks

There are, however, several problems with all of these technologies. Firstly, none natively connect to the internet if you want to implement a smart home system. A Bluetooth device could interact with cloud services through a smartphone when it is in the vicinity but, otherwise, they all require a hardware gateway that connects back to an internet-connected router. Secondly, they are not natively interoperable. You may choose Zigbee as your preferred smart home technology, only to find that a product you'd like to use is only available with Z-Wave. It's possible to develop ways of bridging this gap using Node-RED [5], but this is hardly suitable for most users.

Thirdly, control of the devices on the network can prove unwieldy. Multiple apps may be required, all with different methods of control. Finally, linking voice assistants to these systems can be challenging. Amazon Alexa supports far more devices than does Google Home, which, in turn, is compatible with more devices than Apple HomeKit [6].

To tackle some of these concerns, the technology industry has done what it does best — introduce another technology, Thread.



Figure 3: The Ikea TRÅDFRI range of smart home devices, such as switches and lightbulbs, uses Zigbee.

known as the Bluetooth SIG, and eventually became Bluetooth Low Energy (LE). Today, Bluetooth Classic and LE reside in the same chipset, sharing an antenna and radio transceiver, and have commonalities in their software stack. However, they are not compatible with one another [4].

Despite the low-power performance suited

to battery-powered IoT sensors, it wasn't until 2017 that mesh networking capability for Bluetooth LE became available.

Low-Power Wireless Personal Area Networking Alternatives

By this time, alternative IoT wireless standards were already on the market. Zigbee, an IEEE 802.15.4-based technology,

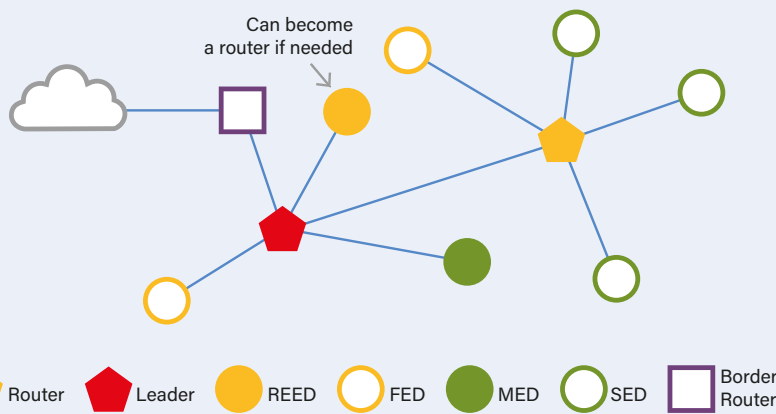


Figure 4: Thread offers a self-healing mesh that, unlike the alternatives, uses the IPv6 standard 6LoWPAN.

Thread Nodes

A Thread mesh network's nodes consist of one of two types of devices [8]: a Full Thread Device (FTD) or a Minimal Thread Device (MTD). MTDs can only be End Devices (ED), requiring a Router node to join the Thread network. A single router can support up to 511 EDs. There are two types of MTD. The Minimal End Device (MED) has an always-on transceiver, while the Sleepy End Device (SED) enables its transceiver to check for messages periodically. SED nodes, such as sensors, will typically be battery-powered.

FTDs always have their transceivers enabled and come in two flavors. The first is Routers, which leave their transceivers powered on to support new nodes joining the networks and forward packets. The first node that takes the role of Router is termed the Leader and manages future nodes added to the network that takes on the role of Router. There can only be one Leader and up to 32 Routers. Should the Leader fail or be removed, one of the other Routers takes on the role. Should the Router role not be required (as there are enough Routers in the mesh), the FTD can become a Router Eligible End Device (REED). It functions as an ED until its Router function is required, at which point it is promoted.

The second is the Full End Device (FED). These FTDs can only function as EDs but keep track of more data about the network than MEDs (such as multicasting and IPv6 address mapping). This profile is suited to mains-powered devices.

With these constituent parts, the Thread mesh is fully operational. However, linking it to cloud services requires one more element — one or more Border Routers. Such devices link the Thread network with either Ethernet or Wi-Fi (Figure 4). They could be integrated into home appliances such as smart TVs, Wi-Fi routers, and voice assistants that are already internet-connected.

The most significant benefit of this is that Thread devices talk secure IP just as all other Ethernet, Wi-Fi, and cellular devices do.

Figure 5: Matter solutions are available from Espressif, as shown in this demonstrator at embedded world 2023.



Figure 6: Silicon Labs also demonstrated Matter operating with both Zigbee and Z-Wave networks.



Tying Everything Together

The Thread Group Alliance was formed in 2014 and has concentrated on building a low-power, low-latency wireless networking technology that competes with the solutions already mentioned. It also operates at 2.4 GHz, but differentiates itself in several ways. Firstly, it uses Internet Protocol (IP) communication — specifically, a version of IPv6 called 6LoWPAN. This adapts IP to the needs of low-power devices and the IEEE 802.15.4 wireless networks used. Mechanisms used include

fragmentation, reassembly and compression that wouldn't be necessary in IPv6.

Secondly, security is more robust and mandatory. Data is encrypted end-to-end, just like in any other TCP/IP network, meaning any devices acting as routers on the network cannot inspect the content of data that passes through them. Finally, compared to the alternatives, mesh latency as data hops across a network is around half that of Zigbee and approximately seven times lower than Bluetooth [7].

By comparison, Zigbee, Z-Wave, and Bluetooth must translate any internet/cloud communication to proprietary wireless networking protocols. But, another key selling point is that the application layer, where the smartphone apps and web interfaces typically attach, is agnostic. This means that, in theory, interoperability between Thread-capable devices and home assistants from different manufacturers should improve.

It also supports yet another new technology — Matter.

Managing Device Interoperability

Matter has been in the works for a few years and was initially known as Project Connected Home over IP, or “CHIP.” One of the contributors is the Connectivity Standards Alliance [9] (CSA), formerly the Zigbee Alliance, which provides certification for Matter products in addition to their existing remit for Zigbee-based devices. As it sits on the TCP/IP transport layer, it is also ready to work with Thread and any Ethernet or Wi-Fi device in the home.


Matter is a software layer that defines how devices communicate with one another, allowing any smart home system or voice assistant to be used. But, perhaps the greatest benefit is Multi-Admin, which allows simultaneous use of different apps with devices. For example, a lightbulb could be controlled by both a wall switch and the app on your smartphone provided by its manufacturer, as well as being connected to and controlled by your chosen voice assistant. This also means that a Google voice assistant should be able to control Amazon or Apple devices. However, there is no guarantee of how far compatibility will extend in reality. As *Wired*’s Simon Hill notes in his article on Matter, there is still the possibility that control of advanced features or settings may require the use of a device or app from the same ecosystem [10].

Several semiconductor manufacturers have demos showing Matter operating with

Thread and Wi-Fi devices, such as Espressif [11] and Silicon Labs [12] (**Figure 5** and **Figure 6**). Currently, the specification supports most basic Smart Home devices, from light bulbs and outlets to locks and thermostats, and the big voice assistant players are in, too. White goods, robot vacuums, and cameras are some items on the list for the next iteration of the specification [13].

Will It Matter?

So, after a decade of Smart Home IoT, we still find ourselves in another VHS-versus-Betamax battle [14] that is getting more complex, not less. The key seems to be Thread, which resolves the long-standing issue with Zigbee, Z-Wave, and Bluetooth Mesh, namely that they require a gateway that translates from IP to the protocol that these wireless networks use. Matter provides the grease that makes Thread make sense, ensuring that users, rather than the consumer electronics giants, can decide which apps and voice assistants they’ll use to control their devices.

It also offers a way to ensure that existing installations don’t become obsolete overnight. Obviously, it can only offer interfacing as far as the gateways of Zigbee and co., not into the networks themselves, but this should help. And, because Matter supports Wi-Fi and Ethernet devices, this could be the factor that makes it a success. Manufacturers of high-value, long-retained consumer goods, such as washing machines and cooking appliances, know that almost all of their customers have Wi-Fi, avoiding having to bet on one of the other three. So, will Matter matter ten years from now? It certainly seems more user-need-centric, so the chances are good. However, as usual, only time will tell. 

230226-01

Questions or Comments?

Do you have technical questions or comments about this article? Email the author at stuart.cording@elektor.com or contact Elektor at editor@elektor.com.

WEB LINKS

- [1] “Microsoft Smart Home,” Microsoft, 1999 [YouTube]: <https://bit.ly/3zoDVCv>
- [2] “There Will Come Soft Rains (short story)” [Wikipedia]: <http://bit.ly/3ZR73x7>
- [3] E. Grabianowski, “Is Wibree going to rival Bluetooth?” HowStuffWorks: <http://bit.ly/3nHHTnb>
- [4] M. Afaneh, “Bluetooth vs. Bluetooth Low Energy: What’s the Difference?” April 2022: <http://bit.ly/3KvmiaA>
- [5] R. Dullier, “Control a Z-Wave plug using a Zigbee button!” March 2021: <http://bit.ly/3nHRdHB>
- [6] M. Timothy, “Amazon Alexa vs. Google Home vs. Apple HomeKit: What’s the Best Smart Home System?” MakeUseOf, March 2023: <http://bit.ly/40C3wDY>
- [7] “Benchmarking Bluetooth Mesh, Thread, and Zigbee Network Performance,” Silicon Labs: <http://bit.ly/3Ge74UF>
- [8] “Node Roles and Types,” Google, February 2023: <http://bit.ly/3m5ZmVN>
- [9] Connectivity Standards Alliance (CSA) website: <http://bit.ly/3nF6qcm>
- [10] S. Hill, “Here’s What the ‘Matter’ Smart Home Standard Is All About,” *Wired*, October 2022: <http://bit.ly/3zrpWf7>
- [11] Espressif’s Solution for Matter: <http://bit.ly/3zsb545>
- [12] Video: Matter over Wi-Fi and Thread Demo - Silicon Labs: <http://bit.ly/3nFZ2gZ>
- [13] J. P. Tuohy, “We’re getting our first look at Matter devices today, and here’s what’s coming next,” *The Verge*, November 2022: <http://bit.ly/410zh9D>
- [14] D. Owen, “The Betamax vs VHS Format War,” *MediaCollege.com*, May 2005: <http://bit.ly/3U78JRD>

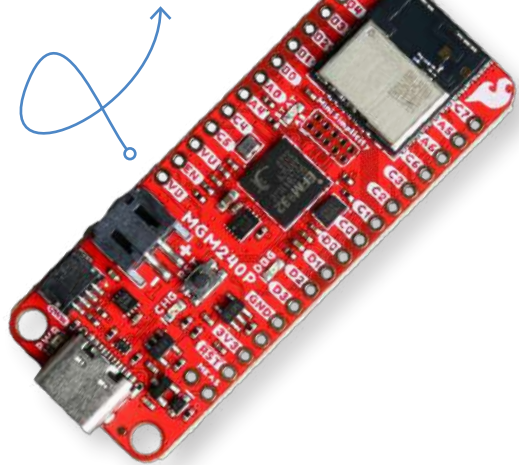
A Matter of Collaboration

Developing with the Thing Plus Matter Board and Simplicity Studio

By Rob Reynolds (SparkFun)

Until now, getting into the home automation game meant choosing an ecosystem. Well, those days are in the past as Matter aims to make it easy for every IoT device to communicate using this new, open-source protocol.

In this article, we are developing a small Matter-compatible demo application, with the new SparkFun Thing Plus Matter Development Board and the Simplicity Studio IDE from Silicon Labs.



Alliance, and a number of other companies such as Nordic Semiconductor, got together in an attempt to develop a single communication protocol that would be used to unify the entire Internet of Things. Matter is an open-source, royalty-free protocol that allows devices to communicate over Wi-Fi, Ethernet, Bluetooth Low Energy, and Thread networks. This means that devices that are Matter-certified will be able to communicate with each other seamlessly, regardless of the wireless technology used.

Now, developers, manufacturers, and consumers no longer have to choose between Apple's HomeKit, Amazon's Alexa, or Google's Weave components. For manufacturers, this means simplified development. For consumers, it means increased compatibility.

One of the key benefits of Matter is that it simplifies the setup and management of smart home devices. With Matter-certified devices, end users will be able to set up their smart home systems quickly and easily, without the need for specialized knowledge or technical skills. And, because safety is of the utmost importance, the protocol also supports end-to-end encryption, ensuring that the data transmitted between devices is secure.

Another key benefit, especially for most of us, is that Matter is completely open source. All of Matter's information is available

By now, we're all well aware of the Internet of Things, or IoT, as it's more commonly called. But, even now, it continues to be rather scattered, with multiple communication protocols. This forces developers, as well as consumers, to decide how they want their devices to communicate, and then locks them into that environment. Those days are coming to an end, with the introduction of Matter, a unified, open-source application-layer connectivity standard built to enable developers and device manufacturers to connect and build reliable and secure ecosystems, and increase compatibility among connected home devices.

A Brief History of Matter

Matter started out as Project CHIP, which stands for Connected Home over IP, in 2019. Major, and previously competing, players such as Amazon, Apple, Google, along with the Zigbee

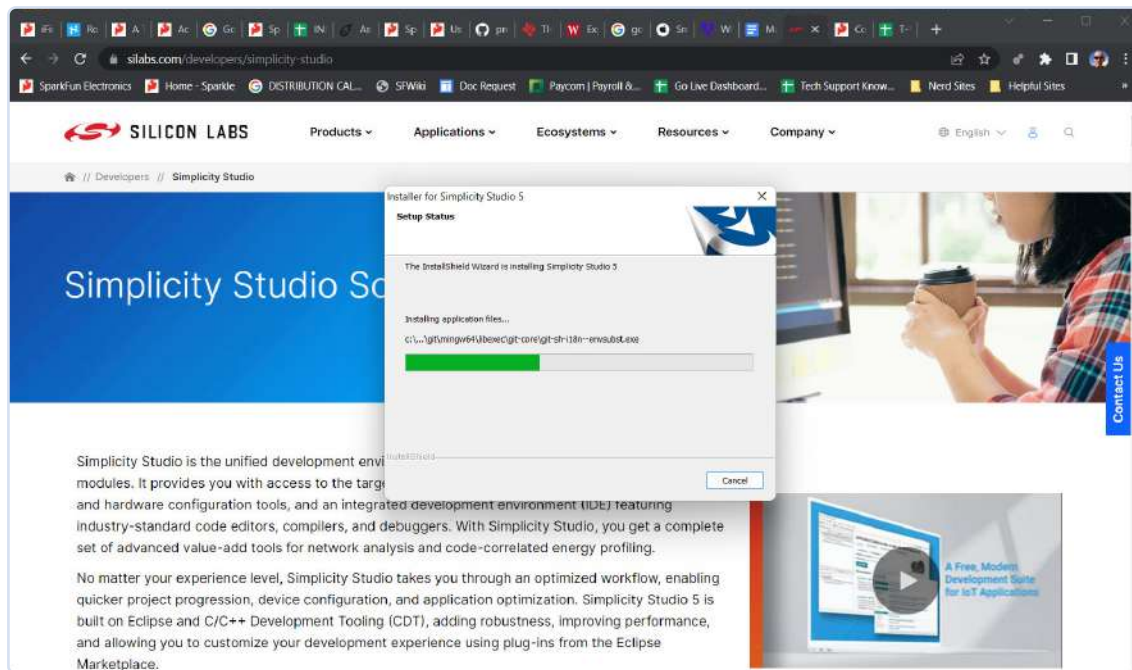


Figure 1: Installing Simplicity Studio.

in their GitHub repository [1], including source code, documentation, scripts, examples, and everything that any developer would need to create Matter-compatible components.

As nice as this is, most of us have still been relegated to the role of consumer, with no way to prototype components without starting completely from scratch. Luckily, that's all changing, with the recent release of the SparkFun Thing Plus Matter Development Board, from SparkFun Electronics [2], which is available in the Elektor Store (see **Related Products**). This is the first easily accessible board of its kind that combines Matter and SparkFun's Qwiic ecosystem for agile development and prototyping of Matter-based IoT devices. The MGM240P wireless module from Silicon Labs provides secure connectivity for both 802.15.4 with Mesh communication (Thread) and Bluetooth Low Energy 5.3 protocols. The module comes ready for integration into Silicon Labs' Matter IoT protocol for home automation. SparkFun's Thing Plus development boards are Feather-compatible and include a Qwiic connector for easy integration into their Qwiic Connect System for solderless I²C circuits.

Getting Set Up with Simplicity Studio

To take our first steps as an IoT developer using Matter, we need to start by connecting the Thing Plus Matter Board to a Google Nest Hub. But, before that, we'll install Simplicity Studio from Silicon Labs. We'll go through the steps quickly here, but you can find a much more in-depth tutorial on SparkFun's website [3].

If you go to the Silicon Labs website, you can download the current version (Simplicity Studio 5 at the time of this writing) for your operating system. When you click the installer button for your particular operating system, you'll be taken to a login page. If you

don't already have an account, create one now, as you'll be asked for it in order to open Simplicity Studio [4]. Once you've downloaded it, run the setup application, as shown in **Figure 1**.

After you've installed and run Simplicity Studio for the first time, you'll be brought to Installation Manager, which will search for any updates available, and, if necessary, list them all for you. Here, you can just hit *Update All* to make sure that you have the latest versions of everything that Simplicity Studio uses to run. If there are no additional updates, or your system is configured to update them automatically, Installation Manager will immediately skip to the next step.

Once Simplicity Studio restarts (if a restart was necessary), you'll be brought back to Installation Manager, but, this time, it will ask if you want to install your device by connecting the device(s), or if we want to install by technology type (wireless, Xpress, MCU, sensors). Here we'll select *Install by connecting devices* (**Figure 2**) and connect our SparkFun Thing Plus Matter board via USB.

You'll be asked if you want to install required packages, which we do, so go ahead and hit *Yes*. Once that installs, the Installation Manager should read "1 Device Found", named something like

☒ J-Link (000449050174) (ID: 000449050174)

Check the checkbox and hit *Next*, and you'll be brought to Package Installation Options. Selecting *Auto* installs all the necessary packages, with all the bells and whistles that we want. There's also an *Advanced* option that lets you choose exactly which packages you do and do not want installed, but, if you're working at that level,

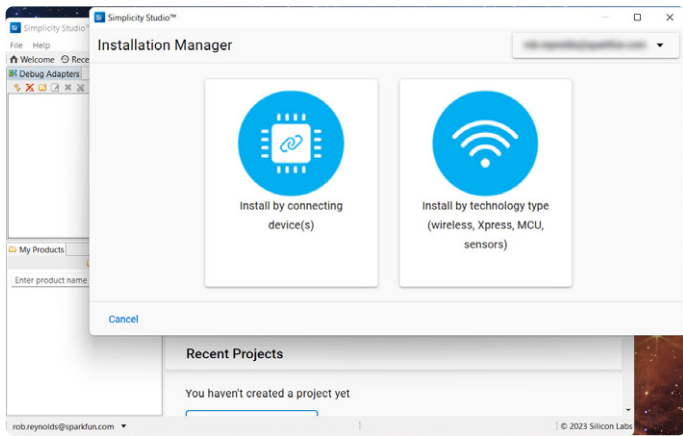


Figure 2: Install by connecting devices in Simplicity Studio.

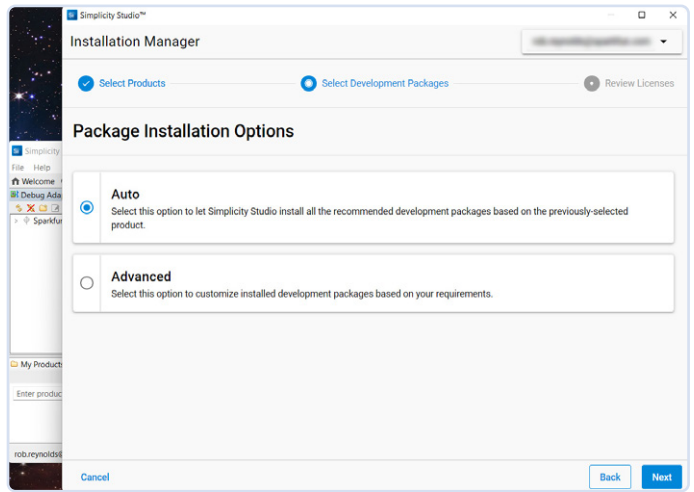


Figure 3: Package Installation Options.

you're probably not reading this article. Go ahead and select **Auto**, then **Next**, as shown in **Figure 3**. You'll be given a Master Software License Agreement to accept, and then the installation will begin. This one will take some time, and requires a restart at the end, so now might be a good time to go get a beverage or a snack.

When you return and restart Simplicity Studio, you'll be greeted by another EULA, and then, finally, we arrive at the *Welcome to Simplicity Studio* screen! Under *Connected Devices*, you should see the SparkFun Thing Plus MGM240P. Hitting **Start** will bring up the Thing Plus information page, where you can see an overview of the board, plus example projects and demos, documentation, and compatible tools. If you navigate to the *Example Projects and Demos* tab, then, in the filter field, type in the keyword "Blink." This should bring up a number of resources. Find the *Platform - Blink Bare-metal* resource, and click the **Create** button (**Figure 4**).

This will open a window that will allow you to change the name or location of the file. I suggest renaming the file to something that makes sense to you, like *FirstBlinkDemo*, then click **Finish**. Once the project is built, you will see on the left side of your Simplicity Studio app a *Project Explorer* window. Look for the main project folder,

which should be named *MatterBlinkExample*. Right-click on that, then navigate to *Run as/1 Silicon Labs ARM Program* (see **Figure 5**).

Clicking on that will compile the sketch and flash it to your Thing Plus Matter board. The board's blue LED should now be blinking at half-second intervals. Now, chances are that when you first connected your Thing Plus board via USB, it started blinking at half-second intervals before you even did anything. To verify that your code is actually being flashed to the board, you can change the blink interval by going to the *blink.c* file in the *Project Explorer*, and changing the interval. Somewhere around line 31 (**Figure 6**), you should find

```
#define TOGGLE_DELAY_MS 500
```

Simply change the value to something that will make it clear that the LED is blinking at a different tempo. Changing it to 100 for an extremely rapid blink, or 3000 for a slow blink, will let you know for certain that the board is being flashed with your code. Once you've changed that value, you can right-click on the *MatterBlinkExample* folder again, navigate down to *Run as/1 Silicon Labs ARM Program*, click that, and watch the board's blue LED change the frequency of its blinks.

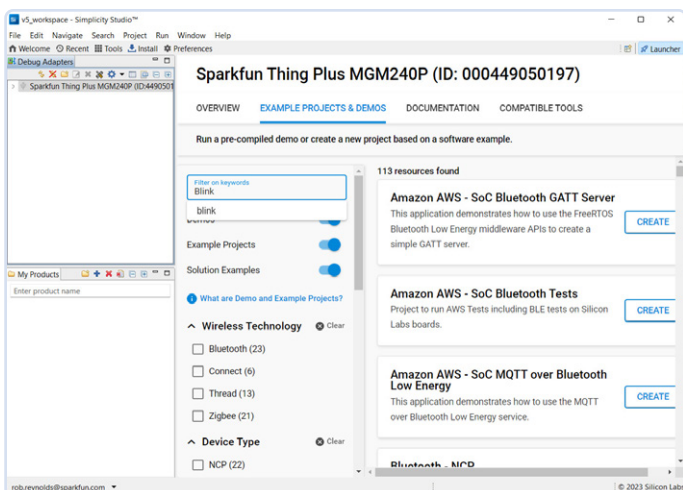


Figure 4: The Example Projects and Demos tab.

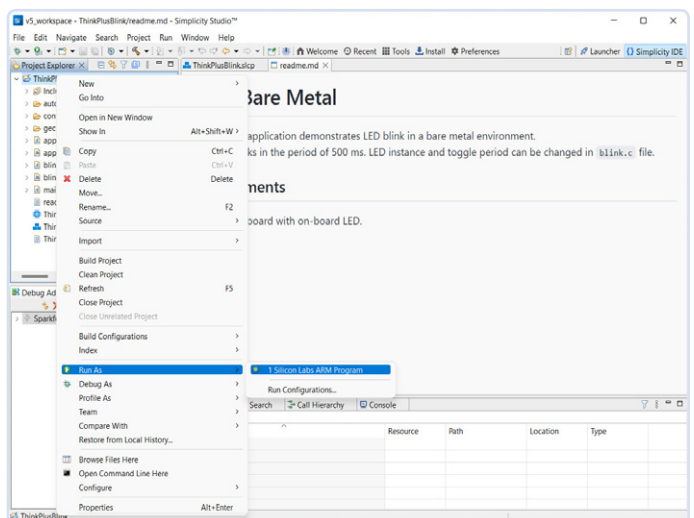
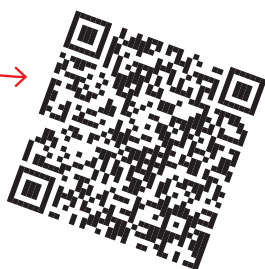


Figure 5: Run the Silicon Labs ARM Program.

Connecting Thing Plus Matter to
Google Nest Hub (Tutorial)
[https://learn.sparkfun.com/tutorials/
connecting-thing-plus-matter-to-google-nest-hub](https://learn.sparkfun.com/tutorials/connecting-thing-plus-matter-to-google-nest-hub)



Connecting the Board to Google Nest Hub

Congratulations, you are now communicating with your Thing Plus Matter board using Matter with the Silicon Labs Simplicity Studio. From here, things really start to get interesting. Communicating with Google Hub allows you to integrate your own custom builds into your Smart Home, with boards such as this SparkFun Thing Plus Matter board (**Figure 7**). If you're excited to keep moving forward, SparkFun Engineer Drew, along with Creative Technologist Mariah, have put together a video and tutorial explaining exactly how to do that [5].

Since this technology is extremely new, examples and tutorials are still fairly limited, but their numbers are growing every day. Getting started now with Matter will put you well ahead of the pack, as this unifying protocol continues across all platforms, and helps to unify the smart home industry. ◀

230224-01

Questions or Comments?

Do you have technical questions or comments about this article? Email SparkFun Support team at support@sparkfun.com, or contact Elektor at editor@elektor.com.



About the Author

Rob Reynolds has been at SparkFun since 2015, and in the role of Creative Technologist for the past five years. With an extensive background in the arts, his experience helps him to create projects, tutorials, and videos that are usually as entertaining and amusing as they are informative. You can find him on Twitter at [@thingsrobmade](https://twitter.com/thingsrobmade).

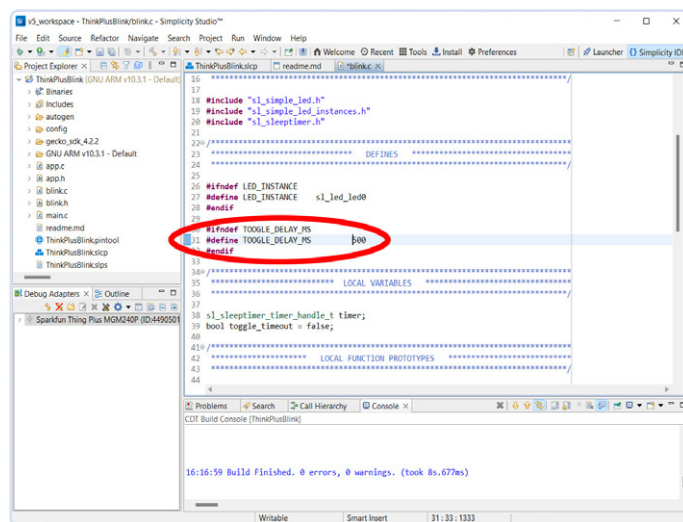


Figure 6: Change the value to make the LED blink at a different tempo.

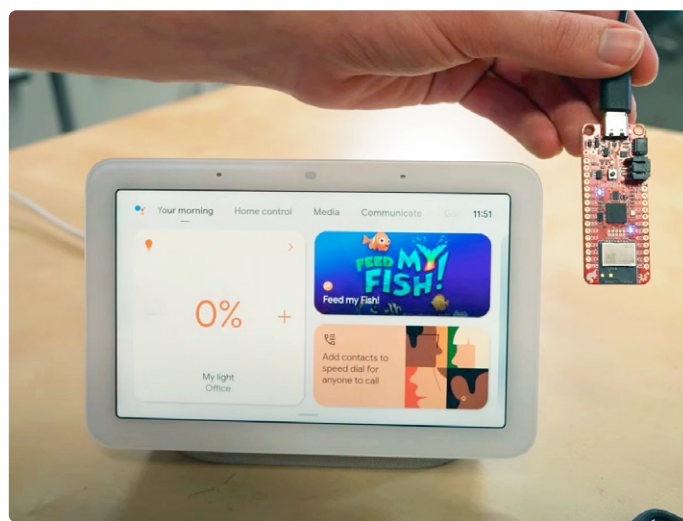


Figure 7: SparkFun Thing Plus Matter and Google Nest Hub.



Related Products

- ▶ **SparkFun Thing Plus Matter (MGM240P)**
<https://elektor.com/20442>

Sparkfun product page
for more information



WEB LINKS

- [1] Matter on GitHub: <https://github.com/project-chip/connectedhomeip>
- [2] SparkFun Thing Plus Matter Development Board: <https://www.sparkfun.com/products/20270>
- [3] MGM240P Hookup Guide: <https://bit.ly/42NRVll>
- [4] Simplicity Studio: <https://silabs.com/developers/simplicity-studio>
- [5] Connecting Thing Plus Matter to Google Nest Hub: <https://bit.ly/3VRcQCI>

The IoT in View

The IoT is anything but old news. In fact, you could argue that IoT innovation is just in its infancy. According to a McKinsey, the total value of the IoT could reach \$12.5 trillion by the year 2030.[1] Sensors are key components to the IoT, and they factor in countless applications

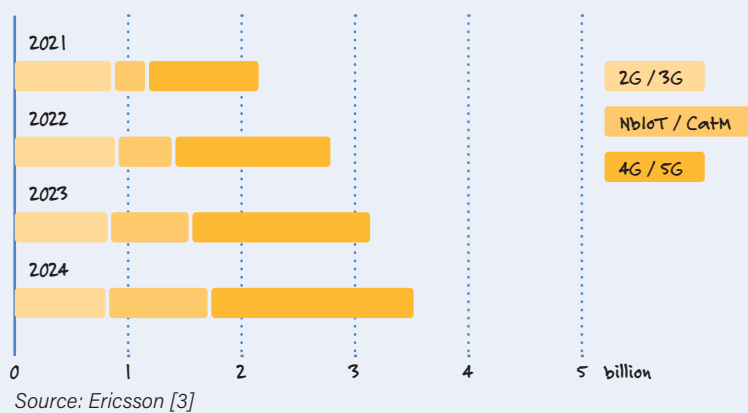
designed and deployed by Elektor community members. The total market of IoT sensors is on pace to increase from around \$10.9 billion in 2022 to \$22.1 billion in 2027,[2] This potential growth signals new opportunities for pro engineers and makers alike.



Five IoT Sensor Tech Trends

- Smart sensors
- Power-efficient sensors
- Soft & virtual sensors
- Sensor fusion
- Biosensors

Cellular IoT Connections by Segment and Technology



\$12.5 Trillion

McKinsey's estimate of the total value of the IoT by 2030.

The Matter Protocol

The matter protocol enables smart home devices to communicate with each other irrespective of their manufacturers. It standardizes devices setup across brands. According to the Connectivity Standards Alliance, "By building upon Internet Protocol (IP), Matter will enable communication across smart home devices, mobile apps, and cloud services, and define a specific set of IP-based networking technologies for device certification." [4] Devices supported by Matter: bridges, controllers, door locks, HVAC controls, lighting and electrical, media devices, safety and security sensors, and window coverings and shades.

37%

Percentage of Internet-connected households owning a smart home device. [6]

68%

Percentage of device owners (or intended owners) who think Matter certification is important. [6]

Devices supported by Matter



Source: CSA [5]

5G Technology

5G — the fifth generation of wireless cellular tech — offers numerous benefits to businesses and consumers: increasingly fast and secure connectivity, lower latency, longer battery life, and more. Since its launch in 2019, it has become one of the most important technologies affecting the industry.

\$ 2 Trillion

The potential boost to global GDP if 5G is deployed across the following key commercial domains: retail, manufacturing, healthcare, and mobility. [8]

50%

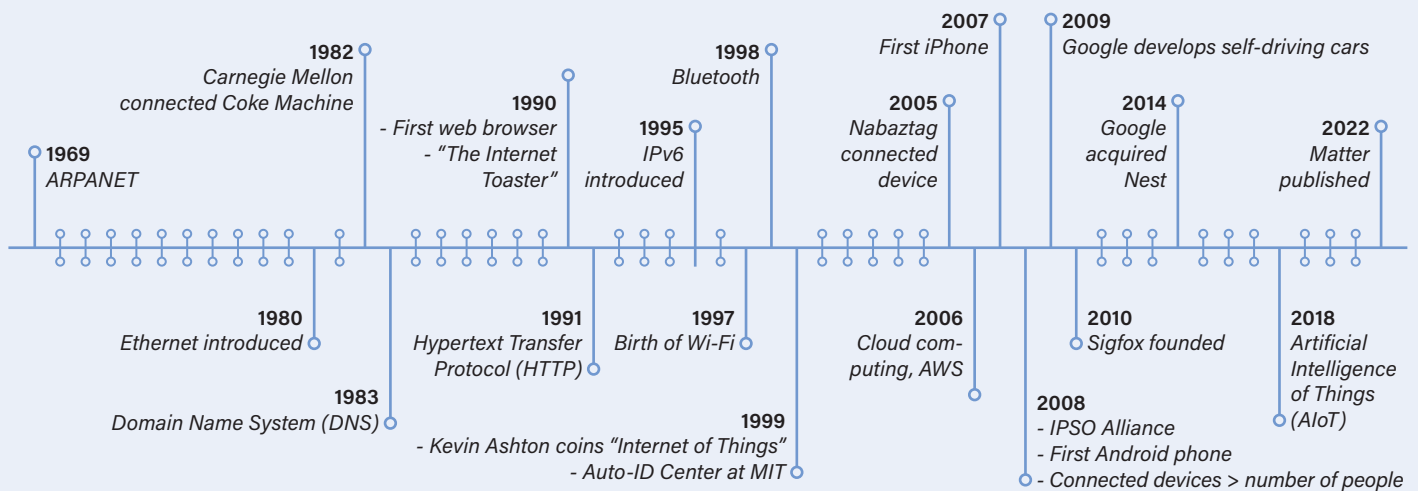
Percentage of respondents to an IEEE survey who said 5G was among the top 5 most important technologies in 2022. [7]

Where do IEEE respondents [7] see 5G playing a role?

- Remote learning
- Telemedicine
- Entertainment
- Day-to-day communications
- Transportation and traffic control
- Manufacturing/assembly
- Energy efficiency

IoT Timeline

Tech pioneer and author Kevin Ashton coined the term “Internet of Things” in the late 1990s. Refer to the following timeline for some key moments in the history of the Internet of Things.



WEB LINKS

- [1] McKinsey & Company, "What is the Internet of Things?" Aug 17, 2022: <https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-the-internet-of-things>
- [2] S. Sinha, "5 IoT sensor technologies to watch," IoT Analytics, Jan 4, 2023: <https://iot-analytics.com/5-iot-sensor-technologies/>
- [3] Ericsson, "IoT Connections Outlook": <https://www.ericsson.com/en/reports-and-papers/mobility-report/dataforecasts/iot-connections-outlook>
- [4] CSA, "Matter: The Foundation for Connected Things": <https://csa-iot.org/all-solutions/matter/>
- [5] CSA, "Matter Executive Overview": <https://csa-iot.org/wp-content/uploads/2022/09/22-Matter-Executive-Overview-One-Pager.pdf>
- [6] C. White, "The Wait Is Over and Matter 1.0 Is Here," Parks Associates, Oct 6, 2022: <https://www.parksassociates.com/blog/article/matter-is-here>
- [7] IEEE, "Advancing Connectivity in 2023," IEEE Transmitter, Oct 24, 2022: <https://transmitter.ieee.org/advancing-connectivity-in-2023/>
- [8] McKinsey & Company, "What Is 5G?" Oct 7, 2022: <https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-5g>



Matter, ExpressLink, Rainmaker — What Is This All About?

Questions by Tam Hanna (Hungary) and Jens Nickel (Elektor)

At embedded world 2023 in Nuremberg, Espressif, well-known for the famous ESP32 microcontroller, showcased a lot of automation solutions that are intended to make the life of IoT app developers (and users) easier. However, it is not easy to understand what is behind new terms such as *ExpressLink* and *RainMaker*. In this interview, Amey Inamdar, Technical Marketing Director at Espressif, shines a light on it. He also answers general questions about his company's portfolio.

Elektor: Please tell us about Espressif today. What is your focus? How has the company changed since its establishment in 2008?

Amey Inamdar: Espressif focused on democratizing the IoT segment with innovative, developer-centric, and affordable Wi-Fi connectivity solutions. We ensured that our hardware was easily accessible and our software available in the open-source community. This is still a core philosophy for Espressif. However, our focus has broadened with changing market requirements.

Espressif is not only continuing to improve Wi-Fi and BLE connectivity by adding Wi-Fi 6, dual-band support, and Bluetooth LE(5) to our portfolio, but also addressing the market's emerging needs by supporting 802.15.4 radio as a foundation for Thread and ZigBee protocols. We've been continuously improving on the power consumption and RF performance aspects of

connectivity. We've also added state-of-the-art interface peripherals to our products.

Additionally, Espressif has been pioneering multi-core MCUs with our ESP32 and ESP32-S3 chips. AI acceleration support in ESP32-S3 is beneficial for ML-on-the-edge applications. Hardware accelerators and media encoders have also been added to newer chipsets.

Furthermore, Espressif SoCs provide security features to ensure that all devices built meet the security requirements. Most of the chips also have innovative security peripherals, such as the digital signature peripheral, which provides integrated hardware secure element-like functionality.

In addition to this, Espressif has also evolved as a complete solution provider, where we identify customer pain points and address them effectively with solutions that go beyond hardware and SDK. ESP RainMaker, ESP Insights, and ESP ZeroCode modules are good examples of this.

Elektor: At embedded world, Espressif showcased home automation solutions, using ESP32 boards connected to AWS. We heard about both "RainMaker" and "ExpressLink." Can you tell us more about these two? Are they independent, or can they play together?

Amey Inamdar: ESP RainMaker is an IoT cloud implementation that you can deploy in your own AWS account. It also has an open-source firmware SDK, phone applications, and voice assistant skills. ESP RainMaker is based on AWS serverless architecture and uses AWS IoT Core and related services internally.

ExpressLink is a connectivity module that provides a simple AT command interface to the host MCU and provides seamless connectivity to AWS IoT

Core and related services, such as OTA. ExpressLink reduces device-side complexity to build and manage connected devices.

ESP RainMaker and ExpressLink are complementary. Customers can use either of them — or both together — to build connected devices with ease.

Elektor: Let us begin with RainMaker [1]. According to the documentation, the RainMaker functions are accessible via ESP-IDF. Since autumn, an Arduino interface has also been available. For professional use, would you still recommend IDF?

Amey Inamdar: ESP-IDF is an SDK for building IoT applications. It is not the only software framework, but ESP-IDF is the project where we first introduce support for new products that we release. ESP-IDF is an open-source project, and it is also a foundation for many other Espressif software frameworks, applications, and solutions.

Arduino provides a simple interface and allows one to take advantage of existing libraries and peripheral drivers. ESP-IDF provides more flexibility for customers who wish to develop multithreaded applications with access to all the native SDK APIs. Customers can choose between the Arduino interface and IDF based on their use cases.

Elektor: Many users seem to like RainMaker, but they're reluctant to host it on AWS cloud. Can Rainmaker also be used without AWS? Is something like this planned for the future?

Amey Inamdar: There are multiple approaches to building IoT cloud platforms. You could build it with Platform as a Service (PaaS) just by using containers or virtual machines from cloud companies. However, such cloud implementations require engineering efforts to maintain scalability and cost. Very few customers can manage such DevOps. Hence, we decided to base ESP RainMaker on the AWS Serverless architecture, which provides a zero-maintenance solution with pay-as-you-go pricing. With this background, you can see that, due to conscious choices made in the favor of customers, it's not easy to use RainMaker without AWS.

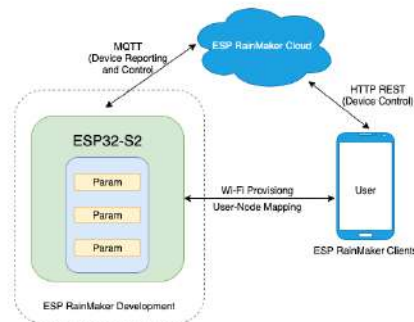
Elektor: The RainMaker specs provide standard types of devices; most of them are in the home automation field (e.g. blinds, fans, and burglar alarms) [2]. However, the system could be used for other applications such as studio lighting and video equipment, not to mention typical industry production equipment. Do you plan to

Get Started

Note: ESP RainMaker works with all variants of ESP32 like ESP32, ESP32-S2, ESP32-C3 and ESP32-S3. We will just use the name ESP32 to mean all of these, unless explicitly specified otherwise.

Introduction

The [ESP RainMaker GitHub project](#) should be used for implementing a "Node" which can then be configured by logged in Users using the clients (like a phone app) and then controlled via the ESP RainMaker Cloud. The examples in this guide are best suited for ESP32-S2-Saola-1, ESP32-C3-DevKitC and ESP32-S3-DevKitC, but the same can be used on other boards by re-configuring Button/LED GPIOs.



broaden the application range, or do you see RainMaker only in the field of home automation?

Amey Inamdar: ESP RainMaker cloud backend is fully agnostic to device types. It acts as a control pass-through and time-series data storage. So, there is no work required to support any particular device type. Even the phone apps provided in the app stores dynamically generate the UI based on the description provided by the device. For example, if the device says that it is a sound mixer that has eight different frequency tuners and that each one represented as a slider from value *min* to *max*, the phone apps will automatically render the UI and no change is required on the cloud side.

Elektor: Wouldn't it be a good idea to open Rainmaker up for other microcontroller companies, for example Microchip?

Amey Inamdar: ESP RainMaker protocol and the device SDK are fully open source, and we don't mind using ESP RainMaker with non-Espressif MCUs.

Elektor: Coming to ExpressLink [3]. We can read on the internet that "ExpressLink-compatible modules provide a simple serial interface through which the host MCU connects to AWS IoT services, thus transforming any offline product into a Cloud-connected one." As we can see on an AWS webpage [4] from Espressif, there seems to be only one compatible board available.

Amey Inamdar: Yes, but this is just a development board for evaluation and prototyping around the ExpressLink compatible module [which is basically an ESP32-C3 module with pre-programmed firmware doing all the AWS magic ~ Ed.]. We have chosen an Arduino-compatible form factor for this development board. The pin layout of this ESP32-C3-AWS-ExpressLink-DevKit is compatible with that of the Arduino Zero

▲
Figure 1: RainMaker connects clients (such as smartphones) and ESP32-based devices via a cloud backend, which is based on AWS.



The smart home industry needed standardization.

Figure 2: RainMaker comes with a powerful SDK and a long list of predefined device types.

ESP RAINMAKER®						
Smart Home Docs API Help						
Get Started	Switch	esp.device.switch	Name, Power*	SWITCH	SWITCH	
Develop Firmware	Lightbulb	esp.device.lightbulb	Name, Power*, Brightness, Color Temperature, Hue, Saturation, Intensity, Light Mode	LIGHT	LIGHT	
Specifications	Light	esp.device.light	Name, Power*, Brightness, Color Temperature, Hue, Saturation, Intensity, Light Mode	LIGHT	LIGHT	X
Services	Fan	esp.device.fan	Name, Power*, Speed, Direction	FAN	FAN	
CLI	Temperature Sensor	esp.device.temperature-sensor	Name, Temperature*	X	TEMPERATURE_SENSOR	
3rd Party Integrations	Outlet	esp.device.outlet	Name, Power*	OUTLET	SMARTPLUG	
Other Features	Plug	esp.device.plug	Name, Power*	OUTLET	SMARTPLUG	X
What's Next?	Socket	esp.device.socket	Name, Power*	OUTLET	SMARTPLUG	X
Documentation Feedback	Lock	esp.device.lock	Name, Lock State*	LOCK	SMARTLOCK	
	Internal Blinds	esp.device.blinds-internal	Name, Blinds Position*	BLINDS	INTERIOR_BLIND	X
	External Blinds	esp.device.blinds-external	Name, Blinds Position*	BLINDS	EXTERIOR_BLIND	X
	Garage Door	esp.device.garage-door	Name, Garage Position*, Lock State	GARAGE	GARAGE_DOOR	X

board and can be directly plugged into it. It also can be easily connected to other host MCUs, such as the Raspberry Pi.

Elektor: Speaking about Matter [5], Espressif is one of the first and most prominent solution providers. How did you get into such a unique position? What made you realize the value of this standard?

Amey Inamdar: The smart home industry needed standardization — previous attempts were unsuccessful. This fragmented user experience made it difficult for consumers to use the connected devices and

difficult for manufacturers to build them. This time, larger ecosystem players came together under the umbrella of the Connectivity Standards Alliance, and showing commitment to make it successful. Furthermore, the design considerations of the protocol have greatly contributed to its success. Notable examples: ensuring cryptographic security for all communication, supporting both Wi-Fi and Thread transports, use of blockchain for authentication and secure OTA, and a few others.

Espressif is in a unique position to offer the most comprehensive solution for Matter with hardware,

Figure 3: ExpressLink is a connectivity module that provides a simple AT command interface to the host MCU.

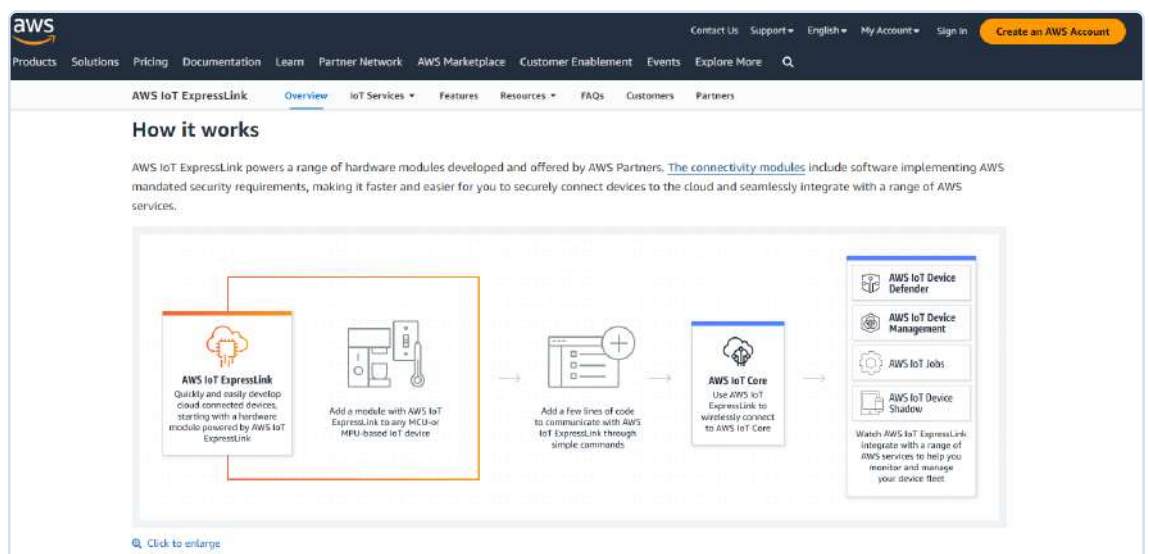


Figure 4: Currently, only a few ExpressLink development boards from Espressif and other companies are available.

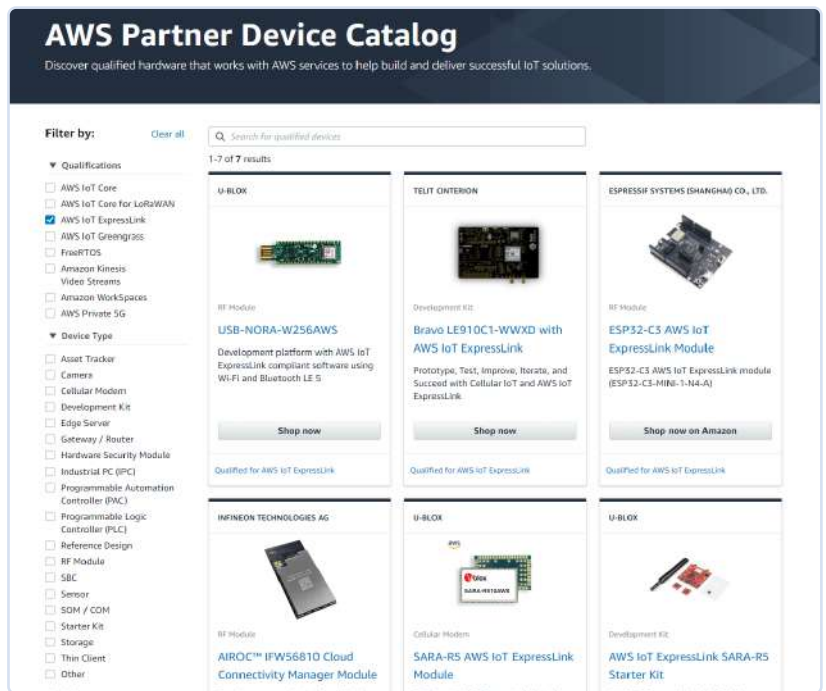
software, ready-to-use solutions, and services. Espressif offers chips and modules to build Wi-Fi Matter accessories, Thread Matter accessories, Thread Border Routers, as well as Matter bridges. *esp-matter* is an open-source SDK that provides tools and examples to create Matter accessories. Espressif's DAC-provisioning service provides modules that are securely provisioned with device attestation certificates so that customers don't have to worry about secure and complex manufacturing. Espressif's certification assistance service helps customers get their Matter-compatible accessories certified. And ESP ZeroCode modules come with pre-certified firmware for simple devices so that customers can directly build Matter-compatible accessories without requiring any development on their side.

Elektor: The ESP32-P4 shows Espressif pushing outside the traditional communications module market. Given that vendors such as ST and Microchip offer their clients automated code generation, what Unique Selling Point do you see for the P4?

Amey Inamdar: ESP32-P4 packs quite a good set of features from better computing power, improved peripherals, and memory architecture. ESP32-P4 can be paired with any other Espressif connectivity chip. When paired, this will unlock some interesting use cases in the higher-end IoT device segment. The key USP of ESP32-P4 will be standardized software support by which the same ESP-IDF will support ESP32-P4, enabling developers to carry their learning and port applications forward seamlessly from other Espressif SoCs to P4. Additionally, the rest of the ecosystem — with high-level language interpreters, RTOS support, SDKs, and software components — will remain usable on ESP32-P4.

Elektor: Given that more and more vendors are pushing into the module space, what do you plan to do to stay on top of the market in the long run?

Amey Inamdar: We want to continue doing the right things for our customers, and we believe that will help us to maintain our leading position, in the long run. I don't think there is a single dimension to



it. A high level of integration in our SoC, innovation, and choice of SoC and communication subsystem architecture, openness in terms of software and information, not compromising on the security features, efficient supply chain, and flexible manufacturing are a few such differentiators. ◀

230227-01

elektor TV
Espressif at the embedded world 2023 show
— watch the Video with Stuart Cording on
the Elektor Industry YouTube channel:
youtu.be/EFnUtAJX2aA



About Amey Inamdar

Amey is working as the Director of Technical Marketing at Espressif. He has 20 years of experience in the area of embedded systems and connected devices with roles in engineering, product management, and technical marketing. He has worked with many customers to build successful connected devices based on Wi-Fi and Bluetooth connectivity.

WEB LINKS

- [1] ESP RainMaker® Webpage: <https://rainmaker.espressif.com/>
- [2] ESP RAINMAKER Standard Predefined Types of Devices: <https://rainmaker.espressif.com/docs/standard-types.html>
- [3] AWS IoT ExpressLink Page: <https://aws.amazon.com/iot-expresslink/>
- [4] ExpressLink Products on AWS Partner Device Catalog: <https://devices.amazonaws.com/search?page=1&sv=iotxplnk>
- [5] Espressif's Solutions for Matter: <https://espressif.com/en/solutions/device-connectivity/esp-matter-solution>



Selecting Microcontroller Dev Kits for IoT and IIoT Applications

An Introductory Guide

By Mark Patrick (Mouser Electronics)

The Internet of Things (IoT) is all around us. For embedded development engineers, embarking on a new IoT design requires strict attention to multiple factors such as power consumption, sensing capabilities, and wireless connectivity. Time-to-market pressures exacerbate the situation. IoT development kits offer a viable and convenient prototyping platform on which to base a design.

However, the capabilities of IoT development kits vary considerably, so careful attention to the application requirements and the kits' features and abilities is needed. This article highlights some of the many considerations involved in selecting an IoT development kit for a new design.

The Online Era

There is no doubting that we are in the online era. Connected devices are all around us. Some we wear, some help us to monitor our electricity consumption accurately, and others notify us if a visitor comes to our door. For industrial production processes, the advent of the Industrial Internet of Things (IIoT) is transforming the way factories operate and helping to increase overall equipment effectiveness. In just a decade, we have changed how we interact and control the world around us. We used to marvel at how we managed without mobile phones, now we have become used to instantly accessing information about aspects of our life and work.

Our cars are experiencing radical changes, too, with up-to-the-minute traffic flow information alarming us of potential delays ahead. Internet-connected health care monitoring equipment allows patients to rest in the comfort of their homes and be reassured that clinical staff as monitoring and on-hand should intervention be required.

The IoT was quickly adopted by industry, emerging as government initiatives such as Industry 4.0 drove the need for automation, process efficiency improvement, and more streamlined operations. An army of sensors



now monitors and reports on the status of each stage of a process, feeding back data to the automation control and analysis system.

The benefits of the IoT/IIoT deployments are considerable, but from an electronic engineering perspective, many challenges are associated with developing an IoT device.

The Requirements of an IoT Device

IoT applications vary considerably, but a core set of functional requirements typically remain the same, whether you are designing a pressure sensor for an industrial process or a room occupancy sensor in an office. The initial fact-finding exercise to establish the outline engineering specification for a new IoT device should consider each aspect highlighted below, since that will shape its functional architecture and design.

Sensing: Sensors sense the world around us, everything from temperature to air pressure to people's movements. For example, a camera might stream data into a machine learning application for object detection, confirming that a label has been affixed to a bottle correctly. Several technical decisions depend on what is being detected and how frequently. Sensor cost, size, and complexity are other considerations. A thermistor used to measure temperature will require additional components for the analogue domain and some software processing before conversion into a digital form. Another factor is how many sensors are needed and how frequently they should be polled.

Connectivity: How will the IoT device interact with a host control system? Is reliable wireless communication available in every use case scenario, or is wired communication preferred? The type of sensor also dictates how much data needs to be transferred and how often. Wireless mesh technology might offer a more robust communication link in a large deployment but requires all IoT devices to operate in this way. For wireless communication, decisions include creating a discrete design or opting for a type-approved module.

Power source: What might be your IoT device's likely power consumption profile? Some applications, communication frequencies, and wireless protocols represent a substantial power load that exceeds the capacity of a small battery. For some deployment scenarios, might a mains (line) supply be available? A recent IoT sensor trend uses energy harvesting technologies to remove the battery altogether. Instead, energy is harvested from ambient energy sources such as solar, vibration, and heat to charge a supercapacitor.

User interface: Will the IoT device require any user interaction? What about during installation and connecting to the host system if not in operation? Is a display or any other form of indication or status LEDs required?

Cloud analysis and control applications: The nature of the IoT is that devices connect to a controlling host system. The connectivity method and protocols will determine the software requirements of the sensor and how it interacts with the host. Is a constant data link required to stream data, or can it be sent at regular intervals as a batch?

IoT Dev Kit Selection

Development kits provide embedded engineers with a convenient and quick way of prototyping a design. In this section of the article, we highlight some of the factors engineers should consider when selecting a suitable kit. There is a wide choice of IoT development and evaluation kits available from leading microcontroller vendors, so it is best to make an informed decision based on the application requirements highlighted above. Below you'll find a list of some features to check when selecting a dev kit platform.

Power supply:

- How is the board powered? USB from a host workstation? Battery? Can it be powered from the intended power source, and does it have a PMIC you could access to try other power sources?
- Is it possible to place a current probe inline to measure real-time power consumption for profiling purposes?

If so, does it include everything on the board and any additional modules, sensors etc.?

Sensors:

- Is the board equipped with the sensor types your application will use?
- Is it possible to add additional sensors? Either using a peripheral connection or an industry-standard add-on format such as mikroBUS Click?
- What peripheral interfaces are available to access? I²C, UART, SPI, GPIO?
- Does the board or microcontroller have an ADC that you could use, and are any additional signal conditioning components required?

Connectivity:

- What wired/wireless connectivity options does the board have? Ethernet, Wi-Fi, LoRa, BLE, ISM etc.
- If no connectivity is on-board, can it be easily added? Does the manufacturer recommend and support a suitable wireless module or is a third-party interface (mikroBUS Click. etc.) option present?
- Is the board's firmware capable of implementing firmware over-the-air updates?

Compute resources:

- Does the board feature the microcontroller that you intend to use? Have you used it before, and do you already have suitable development toolchains?
- Are the board's compute resources adequate enough to run the IoT application, host protocols, and any connectivity protocol stacks?
- If the microcontroller has an integrated wireless transceiver, can you independently control their sleep modes for power-saving purposes?
- What built-in security features does the MCU have and are they suitable for use with your application?

User controls:

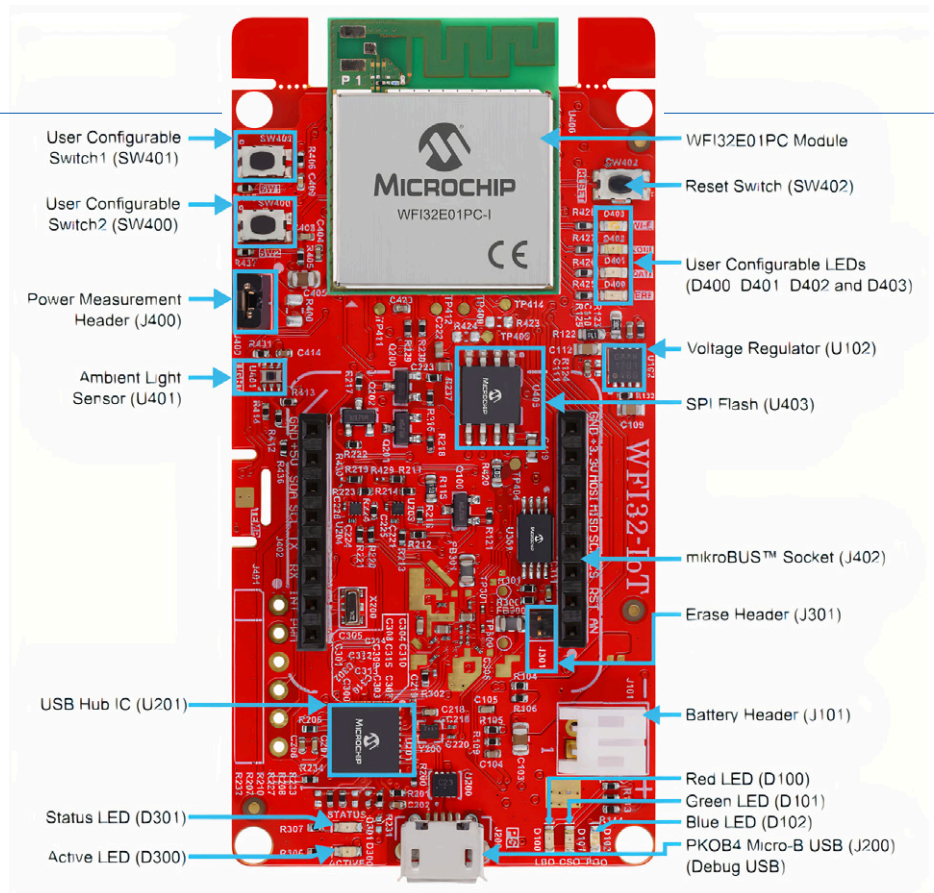
- Is the board equipped with any user buttons, touch-sensing sliders, or other user control hardware features?
- Is a display available? Is it necessary in the end application?

Figure 1: The Microchip EV36W50A IoT development kit (Source: Microchip).

- Are any user LEDs accessible from your code? Are sufficient available, or can you quickly add them using a spare GPIO port?

Software support:

- What is the recommended development toolchain for this board? Do you already have it?
- Is a comprehensive board support (BSP) package included?
- What additional drivers, libraries and firmware are required, and are they royalty-free?
- Check firmware and middleware licensing requirements with the board manufacturer.
- Is the board supplied with a preloaded demo that showcases the board's features? Does it include communication to popular service providers such as Microsoft Azure or Amazon AWS?
- Are other demo and code examples available for the board? Does an ecosystem of library and development partners exist?



IoT Development Board Showcase

Microchip WFI32-IoT development board

The Microchip WFI32, part reference EV36W50A [1], is a comprehensive, fully-integrated, standalone IoT development board (Figure 1). The WFI32-IoT integrates a Microchip WFI32E01PC Wi-Fi 802.11

wireless module based on the PIC family of microcontrollers. On-board sensors include a Microchip digital I²C temperature IC and a digital ambient light IC. Developers can connect additional sensors or peripherals via a mikroBUS socket. The wireless MCU module is also equipped with an integrated antenna. The board can be powered via a

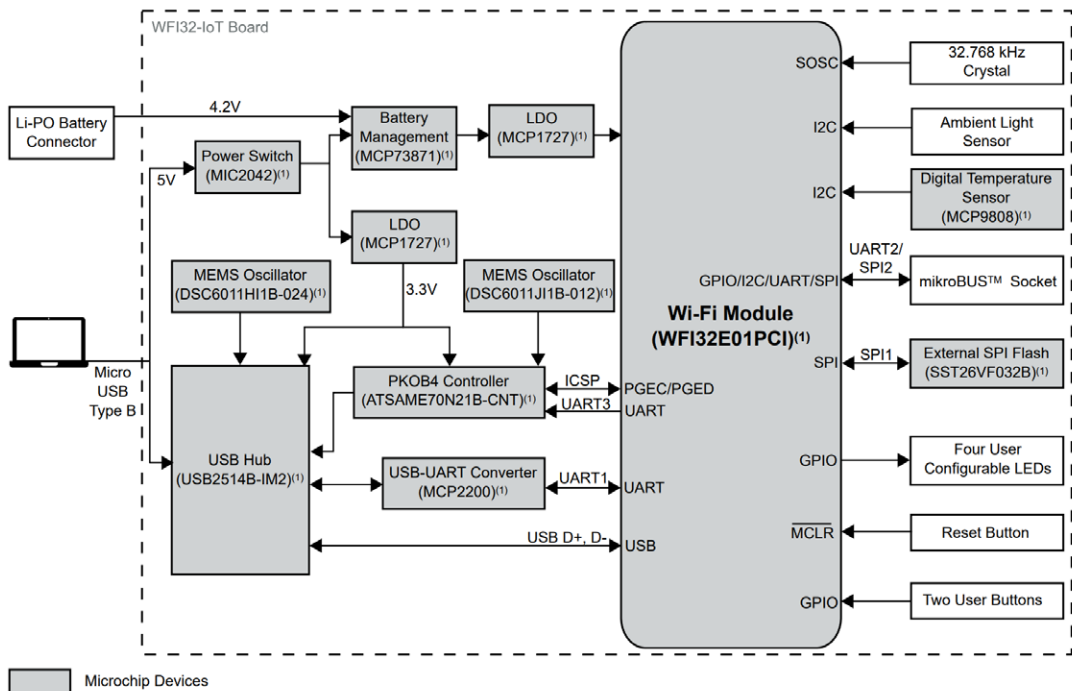


Figure 2: The functional block diagram of the Micro WFI32-IoT EV36W50A development board (Source: Microchip).



Figure 3: The STEVAL-ASTRA1B asset tracking development kit (Source: ST).

workstation host or a LiPo battery. An onboard PMIC provides battery charging capabilities via the USB host.

Figure 2 illustrates the functional block diagram of the WFI32-IoT board and highlights Microchip components integrated into the board. The board is preloaded with an out-of-the-box (OOB) demo image that reads the on-board sensors and sends the data to the Amazon AWS cloud. The demo code and full instructions are available from a GitHub repository [2].

STMicroelectronics STEVAL ASTRA1B multi-connectivity asset tracking reference design

Figure 3 showcases the STEVAL ASTRA1B [3] development kit and reference design. Designed specifically for prototyping and evaluating asset tracking applications, it integrates two wireless connectivity modules. An STM32WB5MMG [4] low-power short-range 2.4 GHz wireless BLE/ZigBee microcontroller module and a long-range sub-GHz STM32WL55JC wireless MCU module for LPWAN communication such as LoRa.

The STEVAL ASTRA1B includes a comprehensive set of sensors capable of measuring multiple environmental and motion parameters. A GNSS module provides outdoor location positioning data. Other board

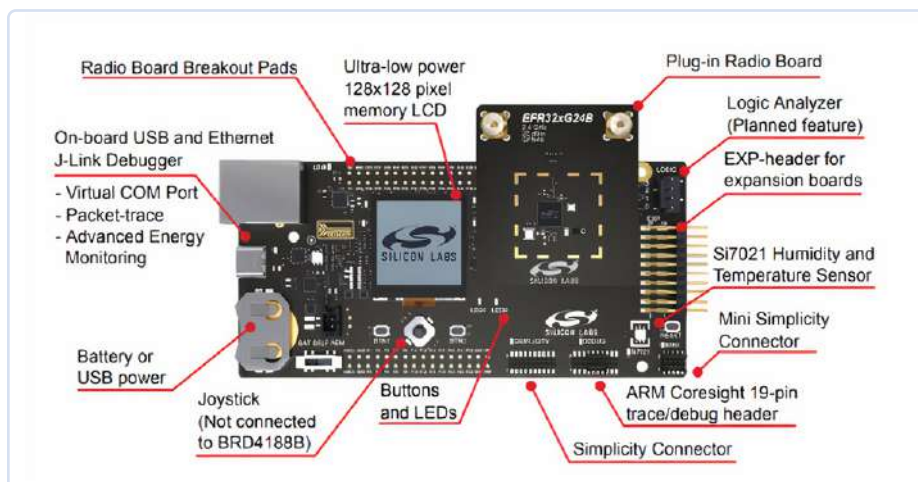


Figure 4: The Silicon Labs xG24-RB4188A antenna diversity module mounted on a Silicon Labs Wireless Kit Pro mainboard (Source: Silicon Labs).

features include a STSAFE secure element, a 480 mAh battery, and an OOB demo consisting of an asset tracking dashboard and smartphone app.

Silicon Labs xG24-RB4188A

The Silicon Labs xG24-RB4188A [5] is a plug-in antenna diversity module for prototyping 2.4 GHz wireless applications (**Figure 4**). It plugs into the Silicon Labs BRD4001 wireless starter board. The module hosts a Silicon Labs EFR32 Wireless Gecko system-on-chip, an RF switch, matching network, and two SMA antenna connectors. RF output from the EFR32 is +20 dBm.

SEMTECH LR1120 development kits

For prototyping LoRa LPWAN applications based on the SEMTECH LR1120 wireless microcontroller [6], SEMTECH offers a range of LR1120 development kits [7], such as the one illustrated in **Figure 5**. The kits are available in regional variants according to the industrial, scientific, and medical (ISM) sub-GHz spectrum. The LR1120 suits multi-regional asset location, inventory management, and theft prevention applications.

Highlighted earlier in this article was the ability to add additional sensors or peripherals to a development board.

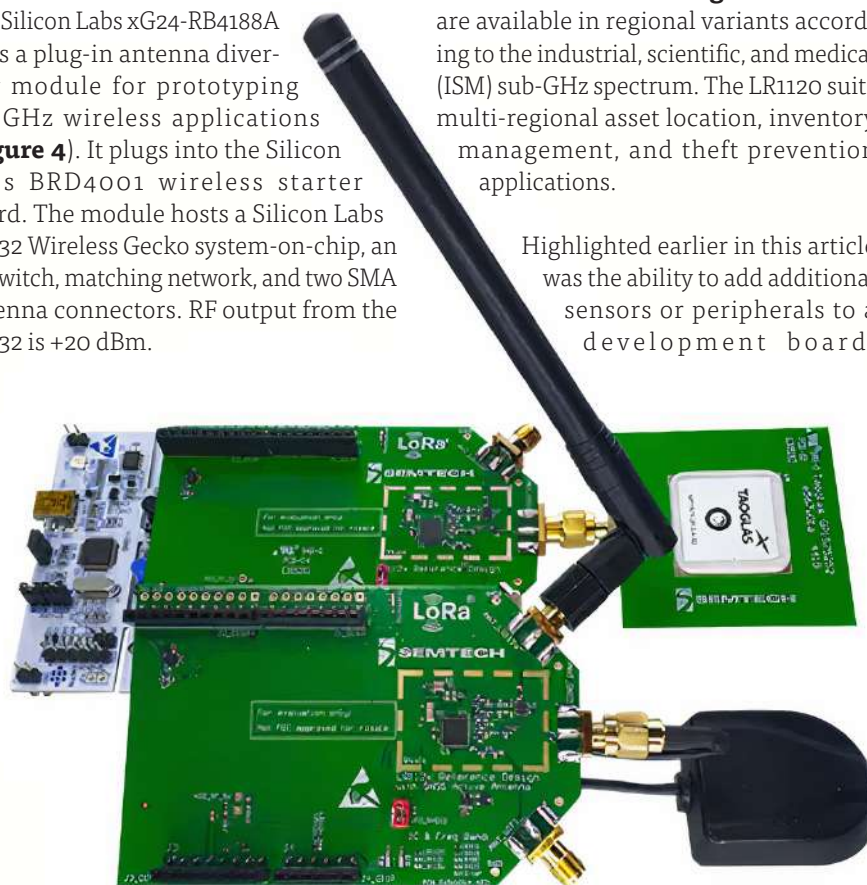


Figure 5: An example of the SEMTECH LR1120 development kits (Source: SEMTECH).



As mentioned in the explanation of the Microchip board, it is equipped with a mikroBUS socket. The mikroBUS, developed by Mikro, has quickly become an industry standard many semiconductor vendors adopt for their development and evaluation boards. mikroBUS brings serial connectivity of SPI, UART, and I²C together with power, analog, and PWM signals to a compact socket format. Mikro has developed hundreds of Click boards [8] that use this convenient form factor.

An example is the Mikro Ultra-Low Press Click [9]. Designed for low-pressure pneumatic measurements, it hosts a TE Connectivity SM8436 pressure sensor

that communicates using the I²C interface (**Figure 6**).

Moving Forward with Your IoT Development Kit

Prototyping an IoT application is made significantly easier thanks to the availability of development boards. This short article highlighted some questions embedded engineers should review when selecting a suitable development board. In addition to the topics mentioned, there will be those specific to the application required that need consideration.

What are you going to develop? ◀

230338-01



About the Author

As Mouser Electronics' Technical Marketing Manager for EMEA, Mark Patrick is responsible for the creation and circulation of technical content within the region — content that is key to Mouser's strategy to support, inform and inspire its engineering audience. Prior to leading the Technical Marketing team, Patrick was part of the EMEA Supplier Marketing team and played a vital role in establishing and developing relationships with key manufacturing partners. In addition to a variety of technical and marketing positions, Patrick's previous roles include eight years at Texas Instruments in Applications Support and Technical Sales. A "hands-on" engineer at heart, with a passion for vintage synthesizers and motorcycles, he thinks nothing of carrying out repairs on either. Patrick holds a first class Honours Degree in Electronics Engineering from Coventry University.

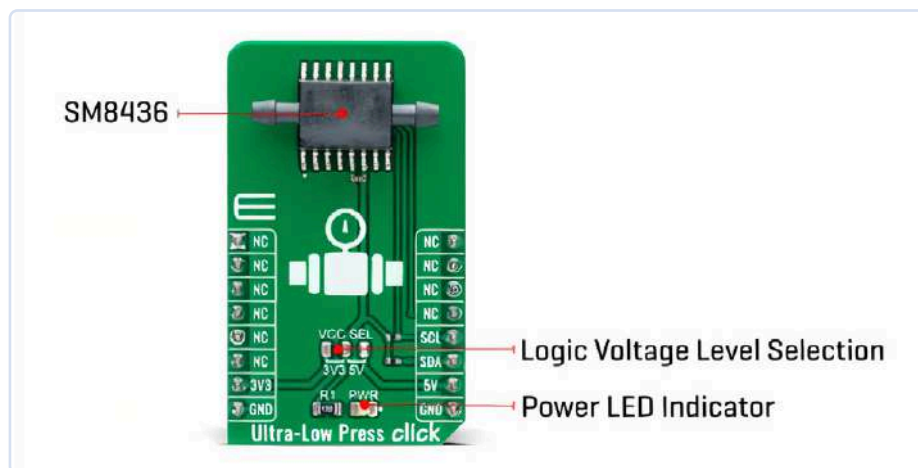


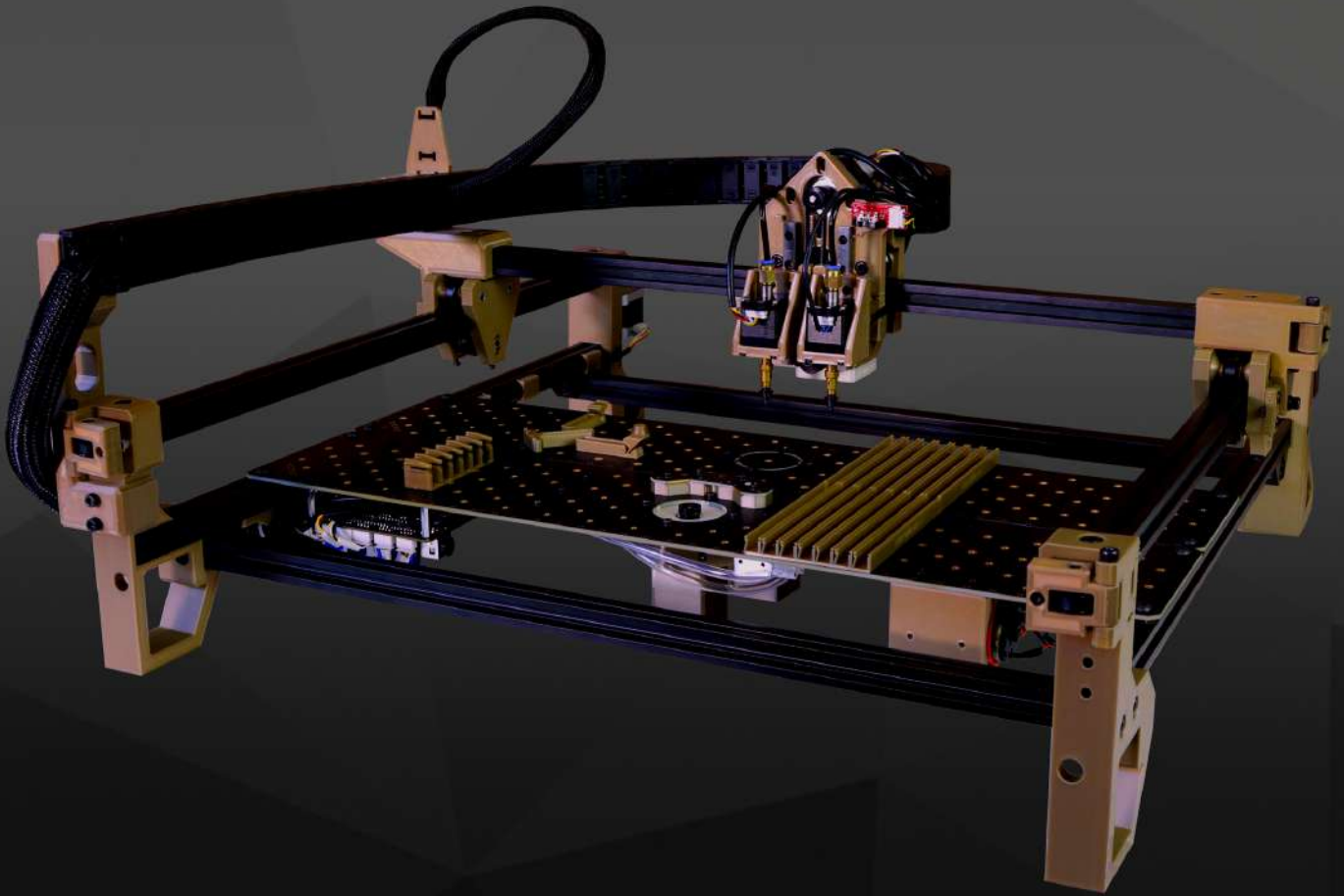
Figure 6: The Mikro Ultra-Low Press Click (Source: Mikro).

WEB LINKS

- [1] Microchip Technology EV36W50A WFI32-IoT Board : <https://bit.ly/3Vw4L5U>
- [2] WFI32-IoT on GitHub: <https://github.com/MicrochipTech/WFI32-IoT>
- [3] STEVAL-ASTRA1B Asset Tracking Evaluation Board: <https://bit.ly/3LwN4P6>
- [4] STM32WB5MMG 2.4GHz Wireless Module: <https://bit.ly/3NFird0>
- [5] Silicon Labs xG24-RB4188A: <https://bit.ly/3LBRFQ5>
- [6] Semtech LR1120 Wi-Fi/GNSS Scanner + LoRa Transceiver : <https://bit.ly/428oET8>
- [7] LR1120 Development Kits: <https://bit.ly/42rdGaO>
- [8] Click Boards™ - Mikro: <https://bit.ly/3LAaH9j>
- [9] Ultra-Low Press Click - Mikro: <https://bit.ly/3LXr2GH>



VS



It's no contest.

The LumenPnP Desktop Pick & Place Machine assembles your boards, so you never need to use tweezers again

- Radically Open Source
- Powered Feeders
- Dual nozzles
- Places 0402s
- Affordable



Opulo.io



Capacitors Do Not Always Behave Capacitively!

By Dr. René Kalbitz (Würth Elektronik eiSos)

Capacitors behave capacitively, at least in the theory of ideal components. However, this is only true under certain operating conditions, and it also depends on the frequency range. This article highlights the impedance spectra of different capacitor technologies and shows when capacitive behavior is to be expected and when it is not.

Impedance and capacitance spectra (or S-parameters) are common representations of the frequency-dependent electrical characteristics of capacitors. The interpretation of such spectra provides a wealth of electrochemical, physical, and technically relevant information. These must be separated from measurement artifacts that are always present, as well as from parasitic effects. Since it is sometimes not possible to map all data in the data sheet, engineers may have to rely on measured

spectra to select the appropriate component for their own circuit design. In order to create the best possible data basis, Würth Elektronik eiSos has implemented the online tool, REDEXPERT [1], in which spectra, but also other measurements, are made available. This article describes the characteristics of such spectra and discusses how basic electrical properties can be derived from them.

Equivalent Circuit for Capacitors

With the circuit shown in **Figure 1**, it is possible to model frequency-dependent impedance spectra for all types of capacitors, from multilayer ceramic chip capacitors (MLCC) to supercapacitors (SC).

C_S stands for the pure capacitance, which does not exist by itself in any electrical component. Every real capacitor has losses that “slow down” the charging process. This phenomenon is described by the pure ohmic equivalent series resistance (ESR). The resistance of the load and the wires also contribute to the ESR.

Pure lossless capacitance C_S is defined by the differential

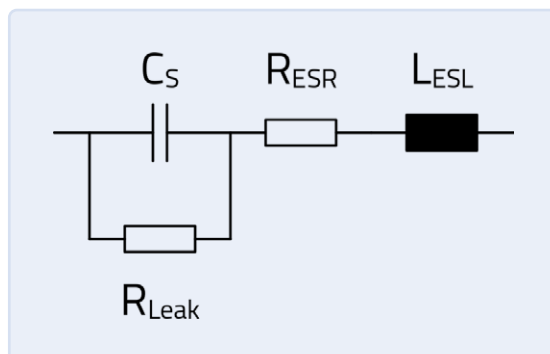
$$C_S = \frac{dQ}{dV}$$

where dQ is the change in charge on the capacitor surface and dV is the change in voltage across the capacitor.

Any alternating current in a metal conductor induces a magnetic field that opposes the current. In the considered model (also L-C-R model or standard model) this property is described by the equivalent series inductance (ESL), represented by L_{ESL} in Figure 1.

C_S , R_{ESR} , and L_{ESL} are the most important parameters required to describe the majority of all spectra. In the simplest approach, they are constants and do not change

Figure 1: Standard equivalent circuit for capacitors, showing capacitance C_S , equivalent series resistance R_{ESR} , equivalent series inductance L_{ESL} , and leakage resistance R_{Leak} .



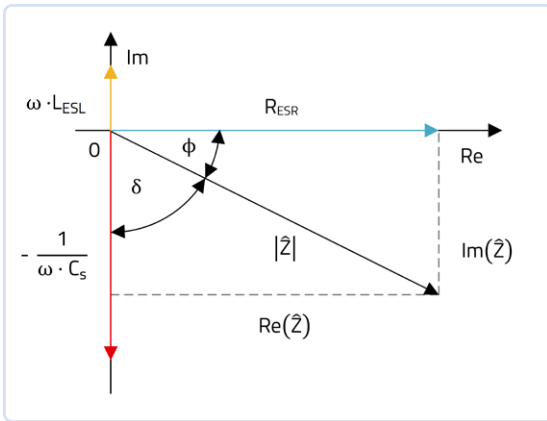


Figure 2: Vector representation of impedance in the complex plane. R_{Leak} is disregarded for simplicity.

with frequency, which is sufficiently accurate for electrical engineering.

The long-term charge loss, i.e., the leakage current, is described to a good approximation by the pure ohmic resistance, R_{Leak} . Usually, R_{Leak} is orders of magnitude larger than R_{ESR} and can often be ignored, i.e., $R_{Leak} \rightarrow \infty$. Its effect is visible in the spectra only at very low frequencies, well below 1 Hz [2].

Impedance and Capacitance Spectra

The following section defines commonly used terms and metrics, such as capacitance and impedance. The above circuit can be described as a frequency-dependent complex impedance \hat{Z} , capacitance \hat{C} , scattering parameter (S-parameter) \hat{S} , dielectric constant ϵ , or any other measurable complex electrical quantity. Impedance $\hat{Z} = \text{Re}(\hat{Z}) + i \cdot \text{Im}(\hat{Z})$ is a complex quantity, with $\text{Re}(\hat{Z})$ and $\text{Im}(\hat{Z})$ as the real and imaginary parts, respectively. Impedance is often expressed by its magnitude, $|\hat{Z}|$, and phase angle, ϕ

$$\hat{Z} = |\hat{Z}| \cdot e^{i\phi}$$

In a complex plane, as shown in **Figure 2**, ϕ describes the angle between $\text{Re}(\hat{Z})$ (abscissa) and complex vector \hat{Z} . Physically, $|\hat{Z}|$ represents the ratio of the voltage amplitude to the current amplitude, while ϕ indicates the phase difference between voltage and current at a given frequency. The phase angle ϕ is related to the loss angle as follows:

$$\arctan\left(\frac{\text{Re}(\hat{Z})}{|\text{Im}(\hat{Z})|}\right) = \delta = \frac{\pi}{2} - \phi$$

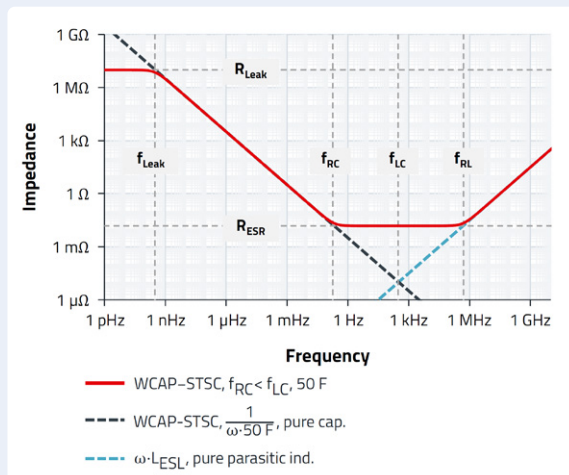
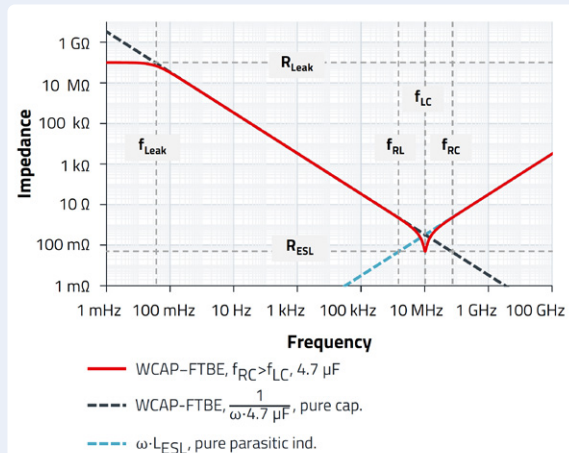
In electrical engineering, it is also common to use the magnitude $|\hat{Z}|$ and its equivalent series resistance $R_{ESR} = \text{Re}(\hat{Z})$. For the example in Figure 1, the equivalent series resistance is the real part of the impedance. To show the relationship between the model and the complex magnitude $|\hat{Z}|$ graphically, all model parameters (except R_{Leak}) are also given in Figure 2. (The mathematical description can be found in the appendix of [2]).

The impedance can also be rewritten into complex capacitance

$$\hat{C} = \frac{1}{i \cdot 2 \cdot \pi \cdot f \cdot \hat{Z}} = \text{Re}(\hat{C}) + i \cdot \text{Im}(\hat{C})$$

All these values, such as $\text{Re}(\hat{Z})$, $\text{Im}(\hat{Z})$, $|\hat{Z}|$, or loss angle δ , can be measured with impedance or network analyzers. Any electronic component (not just capacitors) can be characterized by a set of frequency-dependent values, such as $\text{Re}(\hat{Z})$ and $\text{Im}(\hat{Z})$ or $\text{Re}(\hat{C})$ and $\text{Im}(\hat{C})$. However, it is only through equivalent circuits, such as those shown in Figure 1, that measurement results become interpretable. By adjusting C_s , R_{ESR} , L_{ESL} , and R_{Leak} , it is possible to calculate the basic frequency response for all capacitors. This is shown as an example for the impedance and capacitance spectra of a 4.7 μF and a 50 F capacitor in **Figure 3** and **Figure 4**, respectively. The corresponding phase and loss angles are shown in the appendix of [2].

Figure 3: Impedance spectra $|\hat{Z}|$ for WCAP-FTBE (top) and WCAP-STSC (bottom) as calculated according to the standard model.



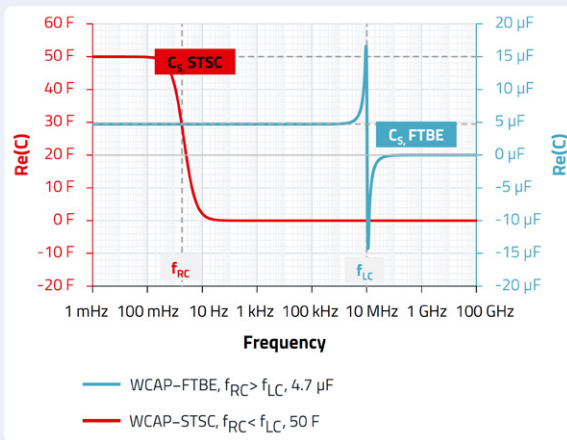


Figure 4: Capacitance spectra $Re(\hat{C})$, as calculated from the standard model. The graph for WCAP-STSC (red) corresponds with the left-hand ordinate and the plot for WCAP-FTBE (blue) corresponds with the right-hand ordinate.

The parameters for the two examples are as follows:

- Supercapacitor (WCAP-STSC) with $C_S = 50$ F, $R_{ESR} = 15$ mΩ, $L_{ESL} = 5$ nH and $R_{Leak} = 10$ MΩ,
- Film capacitor (WCAP-FTBE) with $C_S = 4.7$ μF, $R_{ESR} = 5$ mΩ, $L_{ESL} = 5$ nH and $R_{Leak} = 10$ MΩ.

The parameters were chosen to match the existing WE-eiSos products, which can be found under the product names for film capacitors WCAP-FTBE (4.7 μF) and SCs WCAP-STSC (50 F). In these graphs, C_S , R_{ESR} , L_{ESL} , and R_{Leak} were assumed to be constants and independent of frequency (Table 1).

In general, the position of the most prominent sites in the spectra is described by four characteristic frequencies:

Characteristic frequency f_{RC} of the R_{ESR} -C element:

$$f_{RC} = \frac{1}{2 \cdot \pi \cdot R_{ESR} \cdot C_S}$$

Electrical Parameters	WCAP-FTBE	WCAP-STSC
C_S	4.7 μF	50 F
R_{ESR}	5 mΩ	15 mΩ
L_{ESL}	5 nH	5 nH
R_{Leak}	10 MΩ	10 MΩ

Table 1. Electrical parameters used for the calculation of the spectra.

Characteristic frequency f_{LC} of the L-C element:

$$f_{LC} = \frac{1}{2 \cdot \pi \cdot \sqrt{L_{ESL} \cdot C_S}}$$

Characteristic frequency f_{Leak} of the R_{Leak} -C element:

$$f_{Leak} = \frac{1}{2 \cdot \pi \cdot R_{Leak} \cdot C_S}$$

Characteristic frequency f_{RL} of the R_{ESR} -L element:

$$f_{RL} = \frac{R_{ESR}}{2 \cdot \pi \cdot L_{ESL}}$$

Figure 3 and Figure 4 show two main situations:

Lorentz oscillation: $f_{RC} > f_{LC}$ as in the case of $C_S = 4.7$ μF (blue curve) and

Debye relaxation: $f_{RC} < f_{LC}$ as in the case of $C_S = 50$ F (red curve).

The black and blue dashed lines in both graphs of Figure 3 indicate the purely capacitive and inductive parts. f_{RC} , the characteristic frequency of the RC element, is the frequency at which the capacitor can be charged and discharged. The reciprocal of the frequency is basically the charging time at an ideal constant charging voltage. Above this frequency, the capacitor is no longer fully charged (relative to the maximum voltage of the signal).

At f_{RC} , the capacitance spectrum (Figure 4) of the supercapacitor shows a shoulder. Below this frequency, the capacitance value can be derived from the diagram. Above f_{RC} , the impedance spectrum shown in Figure 3 (below) shows a plateau at R_{ESR} .

The characteristic frequency of the LC element f_{LC} is the frequency at which the coupling of parasitic inductance and capacitance leads to resonant behavior when $f_{RC} > f_{LC}$ as shown in Figure 3 (above). Below this frequency, the capacitor behaves capacitively, i.e., it can store electric charge; above this frequency, the capacitor behaves inductively. The self-resonance leads to a sharp minimum in the impedance spectrum (WCAP-FTBE), as shown in Figure 3 (above). The R_{ESR} value can be read from the minimum of the impedance spectrum. Capacitors should not be operated at f_{LC} or above this frequency in applications.

The capacitance spectrum of the 4.7 μF capacitor, FTBE, in Figure 4, shows one pole. This is a real physical behavior and not just a measurement artifact. The measurement



system, which consists of the capacitor and the parasitic inductor, behaves like an oscillator, or resonant circuit. The physical processes are shown in detail in [2].

f_{Leak} is the characteristic frequency of the R_{Leak} -C element. Below this frequency, the capacitor behaves like a resistor with the value R_{Leak} . Normally, this effect is hardly visible in the spectra. It requires either measurements at frequencies below 1 Hz or a rather low value of R_{Leak} .

The characteristic frequency f_{RL} of the R_{ESR} -L element is the frequency above which the capacitor behaves like an inductor, with the value L_{ESL} (Figure 3, bottom). In cases where $f_{\text{RC}} < f_{\text{LC}}$, this marks the beginning of the rise in impedance at high frequencies.

The two examples discussed here illustrate that a relatively simple model can be used to describe the behavior of high and low capacitance capacitors. The calculated spectra basically show all characteristics that can also be found in measured ones. Certainly, the latter contain further information which requires an extension of the model; nevertheless, the L-C-R model allows us to determine the parameters that are important for the engineering field as well as the basic interpretation of the spectra. The characteristic frequencies presented here

are an important analytical tool in this regard, which can be understood as quasi landmarks for the interpretation of measured spectra. A more detailed consideration of measured spectra of different capacitor types is given in Application Note ANP109 [2].

The consideration includes the following types of capacitors:

- supercapacitors WCAP-STSC
- aluminum electrolytic capacitors WCAP-AIGB
- film capacitors WCAP-FTBE
- multilayer ceramic chip capacitors WCAP-CSGP

230318-01



About the Author

Dr. René Kalbitz studied physics at the University of Potsdam and at the University of Southampton (UK). After completing his diploma, he did research and a doctorate on organic semiconductors and insulators at the University of Potsdam. He gained further experience in applied research at the Fraunhofer Institute for Applied Polymer Research. He joined Würth Elektronik in 2018 as a product manager for supercapacitors and oversees research and development projects in the field of capacitors.



WEB LINKS

[1] REDEXPERT online simulator: <https://redexpert.we-online.com/redexpert/>

[2] René Kalbitz, Impedance spectra of different capacitor technologies. Würth Elektronik AppNote ANP109: we-online.com/en/support/knowledge/application-notes?d=anp109-impedance-spectra-of-different-capacitor-technologies

An NTP Clock with CircuitPython

Why Should You Use This Programming Language?



By Michael Bottin (France)

We take a dive into one of the useful alternatives to Python that's designed to be lightweight enough to work on microcontrollers: CircuitPython. Here, we build a project around a Raspberry Pi Pico W, which will fetch the time from an NTP server and display it on an LCD.

Python is one of the most popular programming languages. As it is an interpreted language, it makes the development phase much faster, unlike languages such as C/C++, which require your program to be compiled before being run. On the other hand, this absence of compilation means an execution environment needs to be installed on the target beforehand.

For over 30 years, Python has been making constant progress in the field of computers and cloud computing. It covers various fields such as scientific calculation, web development (back-end), system scripting, Machine Learning and Big Data (data analysis and visualization). Python is used by many companies, such as Google ("Python where we can, C++ where we must"), YouTube, Instagram, Spotify, Intel, Facebook, Dropbox, Netflix, Pixar and Reddit.

It therefore comes as no surprise that Python made its way into the world of embedded electronics. Yet, to make that

possible, a light version had to be created so that it could run on a microcontroller. But, that version also had to be hardware-oriented in order to use the different peripheral devices available in a microcontroller (GPIO, PWM, I²C, SPI, UART...).

Two versions were created:

- MicroPython, which was created by Damien George in 2013 [1];
- CircuitPython, which was created by Limor Fried (Adafruit CEO), Scott Shawcroft as well as many other contributors [2].

These two languages share the same implementation of the Python programming language (CPython). CircuitPython is an open-source fork of MicroPython. Any MicroPython evolution directly impacts CircuitPython. As far as I know, several articles have already been published on the use of MicroPython in this magazine, but none on the use of CircuitPython.

Advantages of CircuitPython Over MicroPython?

Let's be honest, if you are an IT specialist already familiar with Python and you would like to use this language in the world of embedded electronics, then using CircuitPython would be of no interest to you. In such a case, it would actually be vastly preferable to use MicroPython. However, if you are a maker, a student, or a teacher with barely any knowledge of Python, CircuitPython could be an interesting and suitable choice. Indeed, not only does CircuitPython offer a user-friendly development chain, but it also provides an additional abstraction layer. It is also worth noting that most information about this language can be found online through libraries, tutorials and forums.

That is why I chose to use CircuitPython for this article. I wanted to provide an easy approach to readers wishing to start using Python on microcontrollers. The CircuitPython community is also very active. New versions are released on a regular basis (the current one is the 8.x version) as well as new libraries, and many microcontroller-based boards support this language. CircuitPython can also be run on SBCs such as Raspberry Pi, BeagleBone, Odroid, or Jetson using Blinka API.

Ambition of This Article

This article intends to give the readers a chance to discover CircuitPython and start getting familiar with this language through a simple application.

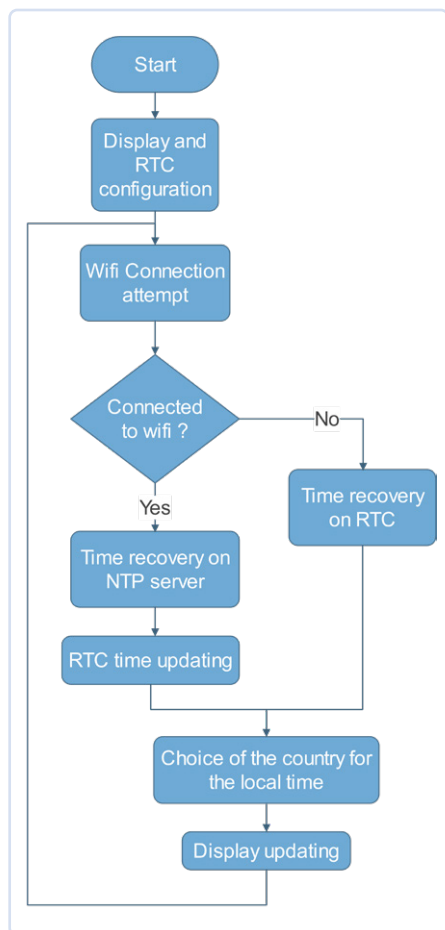


Figure 1: NTP clock general operation flowchart.

I am going to show you how you can create an internet-synchronized NTP clock very easily using CircuitPython. This project also allows you to choose the local time for more than twenty countries around the world. The clock will require a Wi-Fi connection (SSID and password preconfigured) to synchronize the time and a dedicated NTP (Network Time Protocol) server to fetch the timestamp.

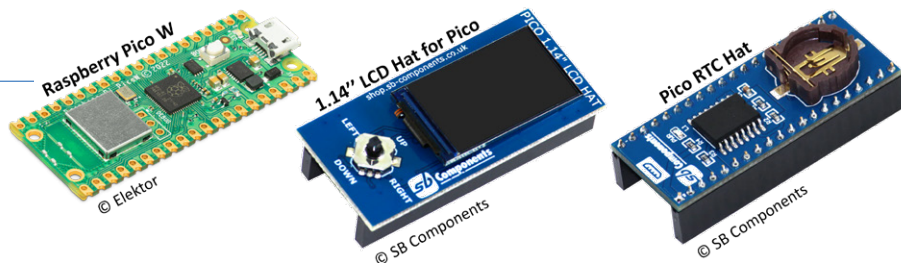


Figure 2: Modules used.

This clock will also be able to keep the time up to date if no network is available, thanks to an embedded real-time clock (RTC), which will save the time when the network is available or will tell time by itself in the absence of network connectivity. A battery will ensure that the RTC is up-to-date when there's no power supply.

The general operation flow chart of the clock is displayed in **Figure 1**.

Presentation of Material

Many development boards (and therefore microcontrollers) can be easily programmed using CircuitPython. I opted for the Raspberry Pico W both because it is a well-known, low-cost board available from most retailers, including Elektor [3], and because it includes a Wi-Fi interface. That being said, it doesn't feature a display or an accurate real-time clock, which means additional components are required for our project.

Instead of creating a dedicated board, I decided to assemble commercial modules for this project. Three stacked modules suffice for this project (see **Figure 2**):

- > a Raspberry Pico W module (be sure to buy the W version, which includes the Wi-Fi interface)
- > a display expansion board module with an LCD screen of 240×135 resolution and a joystick controller for navigation (SB Components [4])
- > a DS3231 real-time clock expansion module (SB Components [5])

Note: The RP2040 microcontroller also features an internal real-time clock that could be used instead of the Pico RTC Hat module. But, as mentioned, the RP2040 is not as accurate and does not feature a backup battery.

Wiring Diagram

Even if we use commercial modules that facilitate the hardware implementation, we need to know the connections between the modules to write the programming code lines of our application. **Figure 3** shows the interconnections between the different modules as well as the names given to these connections, which will be used again in the program. (Note that the SB Components pinout listed in their GitHub documentation is incorrect at the time of writing, but

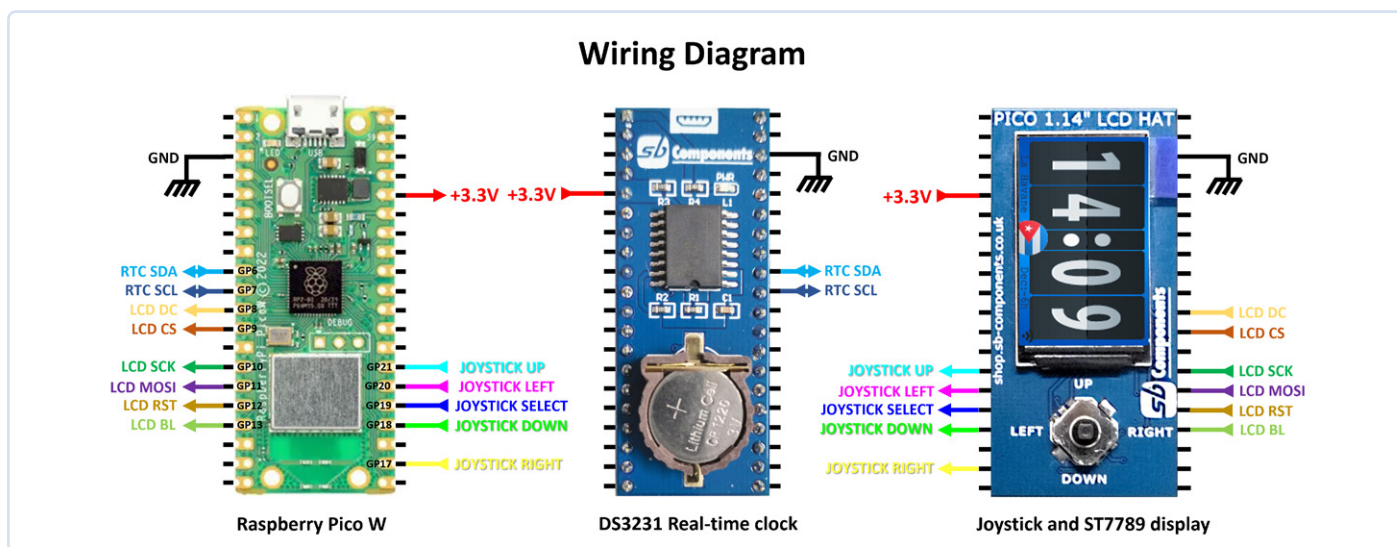


Figure 3: Interconnections between modules.



Figure 4: CircuitPython web page.

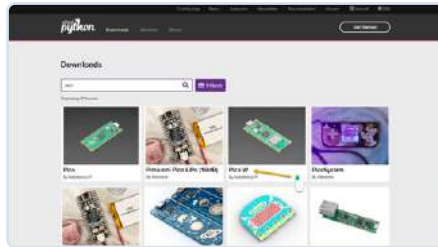


Figure 5: Selecting the Pico W from CircuitPython-compatible boards.



Figure 6: Download the .UF2 file.

their source code refers to the correct pin assignments shown in Figure 3.)

How to Install CircuitPython

The RP2040 microcontroller of the Raspberry Pi Pico W does not feature CircuitPython when you buy the module, so, first of all, you need to “flash” it onto the board.

Just like there are different C Compiler versions according to the target you use, there are as many CircuitPython versions as the number of boards supporting it. As of today, there are more than 380 CircuitPython-compatible commercial boards. Many of the most recent boards from Adafruit are obviously compatible, but many other boards also are, such as boards from Pimoroni, Expressif, Seeedstudio, Waveshare, LilyGo, Cytron, DFRobot and Wiznet.

Most of them use microcontrollers from Microchip (SAM21 / SAM51), Nordic (nRF52840), Expressif (ESP32) or from the Raspberry Pi Foundation (RP2040). You will need to download the corresponding Raspberry Pico W CircuitPython version:

- In your browser, go to the CircuitPython website [6] (**Figure 4**).
- Click on the *Downloads* link and search for the Pico W board (**Figure 5**).
- Download the UF2 file available on the displayed page (**Figure 6**).

The RP2040 microcontroller of the Raspberry Pi Pico already contains a bootloader, which facilitates the installation of CircuitPython onto it. The following instructions will show you how to install the CircuitPython version you have downloaded:

1. Unplug the Raspberry Pico W board from your computer.
2. Press and hold the Raspberry Pico W board's BOOTSEL button while you connect the board to your computer through a Micro USB cable that's compatible with data transfer.

3. A new drive named RPI-RP2 should show up in your file explorer.
4. Upload the downloaded CircuitPython UF2 file to RPI-RP2.

This is all you need to do. Once CircuitPython has been successfully flashed onto the Raspberry Pico W board, a *CIRCUITPY* drive should appear.

How to Use the CIRCUITPY Drive

Here are a few important points to observe while using the *CIRCUITPY* drive:

- This drive works as a USB drive. You can add or delete files/folders from a file explorer. Its capacity is that of the flash memory available on the RP2040 microcontroller to store your code and your resources (images, audio...).
- You can connect the Raspberry Pico W board to your computer without taking any particular precaution. **On the other hand, it is highly recommended to eject it like a USB drive when you want to unplug it, or you run the risk of corrupting the file system.**
- Once you have opened the *boot.txt* file, you can check which version of CircuitPython is installed on the Raspberry Pico W board.
- Once the Raspberry Pico W board is powered through its USB port (either by your computer or by a DC power supply), the CircuitPython code runs. But, only files named *code.py* or *main.py* will run; be sure to name your file *code.py* for this project.
- It is advisable (but not obligatory) to copy the image files into an *./images* directory, the library files into a *./lib* directory, and so on. This will help you to keep an organized tree structure for your projects.

How to Use the IDE

CircuitPython is an interpreted language, so there is no compilation.

If a *code.py* file containing CircuitPython code is present on the *CIRCUITPY* drive, then it will run automatically. That means we could simply edit that file in a text editor and save it before running it, but remember: Program writing often goes hand in hand with debugging. Using an IDE gives you access to tools such as a console to get feedback on errors.

The IDE we are going to use for this project is Mu Editor [7]. It is a free, simple software solution available for Windows, Mac OSX, and Linux. (You can also use Thonny, VS Code, Atom, or PyCharm if you install the CircuitPython extension.)

Figure 7 shows the Mu Editor IDE interface. The toolbar is minimalist:

- *Mode*: Choice of programming language. Make sure to select *CircuitPython* mode.
- *New*: Creation of a new, blank file.
- *Load*: Opens an existing file. If the *CircuitPython* mode is selected and your Raspberry Pico W board is correctly connected, the *CIRCUITPY* drive folder should appear automatically in the dialog window.
- *Save*: Saves the current file (the active tab). A small red dot next to the filename in the tab indicates that the file has been modified since the last backup.
- *Serial*: Displays/hides the console. I recommend that you always display the console (REPL) as it will display error messages and what you chose to print from the code running on the Pico.
- *Plotter*: Displays/hides a scrolling graph showing digital data which your code may create.
- *Zoom-in* and *Zoom-out*: Makes the font smaller or bigger.
- *Theme*: Toggles between three different display themes: *Day* (light theme), *Night* (dark theme) and *High-Contrast*.
- *Check*: Searches for code errors, even if they have no impact on its running.
- *Tidy*: Tidies your code by deleting useless spaces or useless blank lines, for example.



Figure 7: Mu Editor IDE interface.

- **Help:** Online help.
- **Quit:** Quits the editor.

So, here are the different steps to follow to develop a program in the IDE:

- › Open the `code.py` file from the CIRCUITPY drive.
- › Modify the code in the editing area.
- › Save the changes to your file.
- › Observe the results of the execution in the Serial console. Go back to Step 2 if any error occurred during the execution.

You can find more information at the following website links:

- › Start Here! (about the interface) [8]
- › What is a REPL? (the console) [9]

- › Plotting Data with Mu [10]
- › Keyboard Shortcuts [11]

Now that you've got all you need to get your project started, let's get to the heart of the matter: the NTP clock.

Wi-Fi Connection Test

CircuitPython supports native Wi-Fi on the Raspberry Pico W board, so you don't need to install any additional libraries. To connect to Wi-Fi, you must provide your network's SSID and its password. To prevent this information being embedded directly

in the code, CircuitPython makes use of environment variables.

A `settings.toml` file [12] is located at the root of CIRCUITPY and this data will remain "secret." Any confidential data, such as API keys, could be stored in this file, but in our case it will only contain the SSID and password of the network you want to connect your Raspberry Pico W board to (see **Figure 8**). (Note: This file cannot be edited in the IDE; you have to use a text editor to create it and complete it.)

The code to connect to the network is displayed in **Listing 1**. You can see at first glance that the code is very compact:

- › **Line 2:** Import of the `os` library, which allows you to read the environment variables of the `settings.toml` file.
- › **Line 4:** Import of the `wifi` library, which enables network connection.
- › **Lines 6–7, 12:** These lines are not

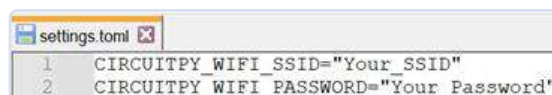


Figure 8: SSID and password in `settings.toml`.



Listing 1: Wi-Fi connection code.

```
1 # CircuitPython's own libraries
2 import os
3 import ipaddress
4 import wifi
5
6 print()
7 print("Connecting to Wi-Fi")
8
9 # connect to your Wi-Fi network with SSID/PASSWORD in 'settings.toml'
10 wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'), os.getenv('CIRCUITPY_WIFI_PASSWORD'))
11
12 print("Connected to Wi-Fi")
13
14 # pings Google DNS server
15 ipv4 = ipaddress.ip_address("8.8.8.8")
16 print("Ping google.com: %f ms" % (wifi.radio.ping(ipv4)*1000))
```

Figure 9: Downloading the libraries (the date in the file name will obviously be different).

necessary, but they provide information to the user through the Serial console.

- **Line 10:** This one line is enough to create the physical network connection.
- **Lines 3, 14–16:** These lines are not necessary, but they enable you to run a test (ping) by querying a server and therefore to check your Wi-Fi connection status, then output the result to Serial Console.

Adding Useful Libraries

Most of the basic libraries can be found on CircuitPython, rather than all of the available libraries, otherwise it would unnecessarily overload the microcontroller's Flash memory. So, it is necessary, depending on the projects you are working on, to add a few libraries.

Installing additional libraries in CircuitPython simply consists of copying the corresponding files to the *CIRCUITPY* drive. So that you only have to do it once, install all the libraries that may be useful for the final version of the project.

Here is the procedure to install them:

1. If it does not already exist, create a *./lib* folder on your *CIRCUITPY* drive.
2. All the available libraries can be downloaded as a single archive from the CircuitPython website (Note: they are also individually available on GitHub)
3. Once on the website, click on the *Libraries* link.
4. Download the bundle that matches the version of your CircuitPython: Version 8.x in our case (see **Figure 9**).
5. Unzip it wherever you want on your computer.
6. In the unzipped bundle folder, open the *lib* folder.
7. Select:
 - a. These folders:
 - i. *adafruit_register*
 - ii. *adafruit_imageload*
 - iii. *adafruit_display_text*
 - b. These files:
 - i. *adafruit_debouncer.mpy*
 - ii. *adafruit_ds3231.mpy*

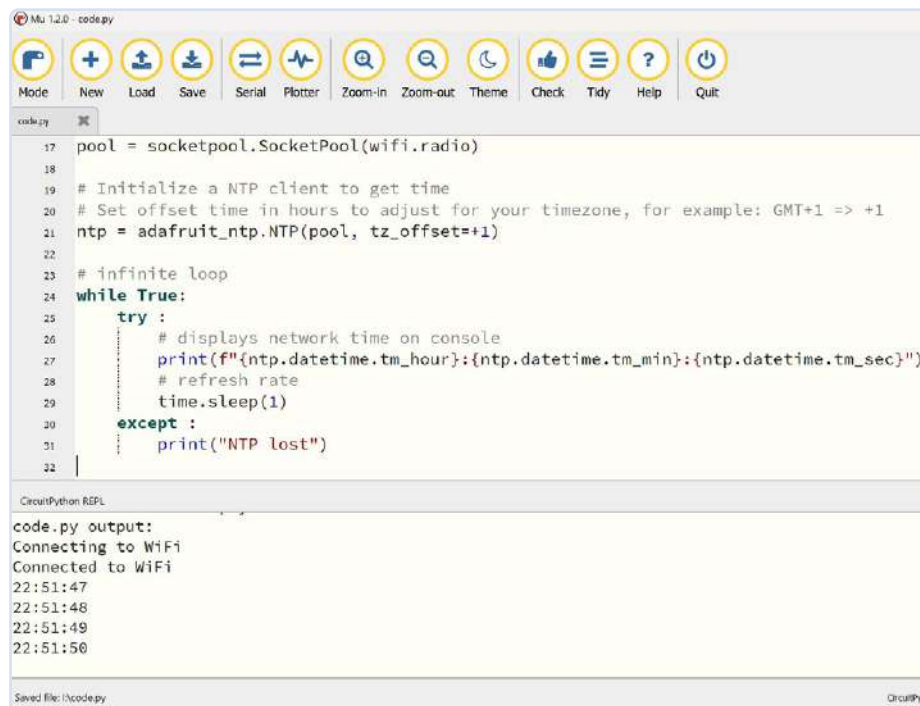


Figure 10: NTP clock displayed in Serial Console.

- iii. *adafruit_ntp.mpy*
- iv. *adafruit_st7789.mpy*
- v. *adafruit_ticks.mpy*

8. Copy the selection to the *./lib* folder on the *CIRCUITPY* drive.

Note: Circup is a tool written in Python that can check the version of your libraries, update them and install their dependencies [13].

Time Display in Console

In the previous code, you connected your Raspberry Pico W board to the Wi-Fi network. Now, you're going to query a dedicated NTP server so that it gives you the time and date. The code that will allow you to fetch the time and display it in the console (**Figure 10**) is available in **Listing 2**.

Let's have a look at the added lines:

- **Lines 5 and 18:** Import of the *socketpool* native library and creation of a socket that will maintain the client-server connection.
- **Line 22:** Instantiation of a NTP client through the socket. The *tz_offset*

parameter is used to set the time difference between UTC (Coordinated Universal Time [14]) and your own time zone. In France, we use CET, which is basically UTC+1, hence the +1 value. You have to adjust it for your time zone. The default server is the Adafruit server (but it can be modified) and the default timeout is 10 seconds.

- **Lines 24–32:** In an infinite loop, the time is displayed in the console according to the desired refresh rate (in our case, 1 second — line 29). To achieve this, we use the *ntp.datetime* parameter. This parameter returns a named tuple from the *time* class containing the date and time information from the server. We access each element of the tuple through its named field [15]. We use the *try...except* construct to intercept an exception that would mean data has not been received once the timeout has been reached.

Time Display on LCD Screen

For this step, we are going to use the screen instead of the console to display the time. CircuitPython features a native library



Listing 2: NTP request code.

```

1 # CircuitPython own's libraries
2 import time
3 import os
4 import wifi
5 import socketpool
6
7 # CircuitPython external libraries
8 import adafruit_ntp
9
10 print("Connecting to WiFi")
11
12 # connect to your WiFi network with SSID/PASSWORD in 'settings.toml'
13 wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'), os.getenv('CIRCUITPY_WIFI_PASSWORD'))
14
15 print("Connected to WiFi")
16
17 # Setting up a socket
18 pool = socketpool.SocketPool(wifi.radio)
19
20 # Initialize a NTP client to get time
21 # Set offset time in hours to adjust for your timezone, for example: GMT+1 => +1
22 ntp = adafruit_ntp.NTP(pool, tz_offset=+1)
23
24 # infinite loop
25 while True:
26     try :
27         # displays network time on console
28         print(f"::")
29         # refresh rate
30         time.sleep(1)
31     except :
32         print("NTP lost")

```



Figure 11: NTP clock displayed on LCD.

called *displayio*. This library provides methods and common attributes for every screen supported by CircuitPython.

The color LCD module we are using offers the following features:

- > SPI protocol
- > 240 × 135 pixels resolution
- > ST7789 driver

You can download the code from the link at the end of the article. It would take a long time to explain in detail, but here are the main steps to reach the same result, as shown in **Figure 11**. You must import several libraries:

1. CircuitPython native libraries:
 - board*: to access the names of the Raspberry Pico W pins
 - displayio*: to manage the graphics
 - busio*: for the SPI bus communication

terminalio: to get a font for the screen

2. External libraries:

adafruit_st7789: to manage the LCD screen

adafruit_display_text: to display the text area on the screen

Then, you must configure the display settings:

1. Create the SPI communication bus *SPI_bus*.
2. Create the bus *display_bus* to connect the Raspberry Pi Pico W board to the LCD. It contains the SPI bus, and also two additional signals (*Command* and *chip_select*).
3. Create the screen *display* by providing the previous bus, by defining its resolution (*width=135*, *height=240*) and by adjusting the offsets in x (*rowstart=40*) and y (*colstart=53*). The values of those

two last parameters can be explained by the fact that the highest resolution of the ST7789 driver is of 320 x 240 while the resolution of our screen is only of 240 x 135 (see **Figure 12**).

4. In CircuitPython, any graphic element must belong to a group. Create a group named *display_group* and create a *time_label* text area. Add the text area to the group. Finally, display the group on the screen.
5. To update the time on the screen, you just need to modify in the infinite loop, the *text* attribute from our *time_label* text area object by copying the string of characters that we used to display in the console.

Real-Time Clock Test

Our web synchronized clock works perfectly fine, so shall we call it a day, then? No, we are not done yet, because what would



Listing 3: RTC code.

```
1 # SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
2 # SPDX-License-Identifier: MIT
3
4 # Simple demo of reading and writing the time for the DS3231 real-time clock.
5 # Change the if False to if True below to set the time, otherwise it will just
6 # print the current date and time every second. Notice also comments to adjust
7 # for working with hardware vs. software I2C.
8
9 import time
10 import board
11 import busio
12 import adafruit_ds3231
13
14 i2c = busio.I2C(scl=board.GP7, sda=board.GP6)
15 rtc = adafruit_ds3231.DS3231(i2c)
16
17 # Lookup table for names of days (nicer printing).
18 days = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday")
19
20 # pylint: disable-msg=using-constant-test
21 if True: # change to True if you want to set the time!
22     # year, mon, date, hour, min, sec, wday, yday, isdst
23     t = time.struct_time((2023, 03, 09, 14, 58, 15, 3, -1, -1))
24     # you must set year, mon, date, hour, min, sec and weekday
25     # yearday is not supported, isdst can be set but we don't do anything with it at this time
26     print("Setting time to:", t) # uncomment for debugging
27     rtc.datetime = t
28     print()
29 # pylint: enable-msg=using-constant-test
30
31 # Main loop:
32 while True:
33     t = rtc.datetime
34     # print(t) # uncomment for debugging
35     print(
36         "The date is {} {}/{} / {}".format(
37             days[int(t.tm_wday)], t.tm_mday, t.tm_mon, t.tm_year
38         )
39     )
40     print("The time is {}:02:{}".format(t.tm_hour, t.tm_min, t.tm_sec))
41     time.sleep(1) # wait a second
```

happen if the network connection was interrupted? Or if the power was temporarily disconnected?

To avoid losing the current time, we are going to add a real-time clock. Coupled with a backup battery (CR1220), it will keep the time updated for years to come, even when the clock is not powered through the USB micro port. This RTC uses the well-known DS3231 circuit, and that circuit uses the I²C bus to communicate with Raspberry Pi Pico W.

Before combining the RTC with the NTP server, you are going to test it on its own and, to achieve that, you're going to need the *adafruit_ds3231* library. Like most of CircuitPython libraries, it offers online tutorials and examples [16].

In the code proposed in this tutorial, you just need to modify the first lines as shown in **Listing 3**. Indeed, many Raspberry Pi Pico Ws do not support the I²C protocol [17]. You need to know exactly which ones are physically connected to the DS3231 RTC

component (see Figure 3). During execution, you should obtain a similar result to that shown in **Figure 13**.

Final Code

The primary function of our project has been achieved. You can find the link to download the final code at [24]. **Figure 14** shows the general interface of the screen display. You can notice that the graphic aspect has been improved and that new elements have appeared on the screen. Once again, a detailed explanation would

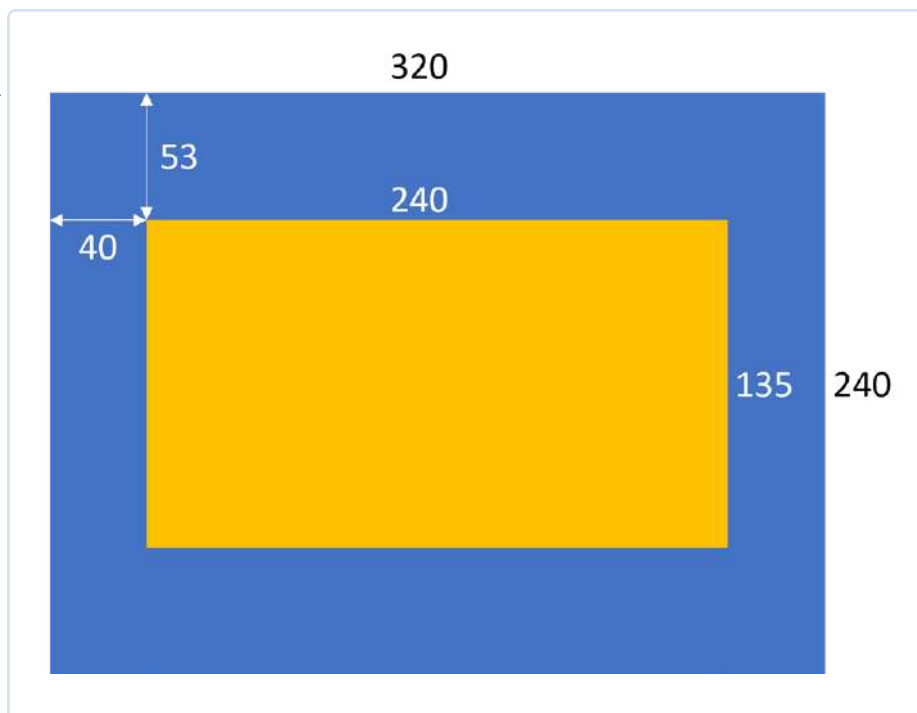


Figure 12: ST7789 display offsets.

```

24 t = time.struct_time((2023, 03, 09, 14, 58, 15, 3, -1))
25 # you must set year, mon, date, hour, min, sec and weekday
26 # year day is not supported, isdst can be set but we don't do anything with it at this time
27 print("Setting time to:", t) # uncomment for debugging
28 rtc.datetime = t
29 print()
30 # pylint: enable-msg=using-constant-test
31
32 # Main loop:
33 while True:
34     t = rtc.datetime
35     # print(t) # uncomment for debugging
36     print(
37         "The date is {} {}/{}{}".format(
38             days[int(t.tm_wday)], t.tm_mday, t.tm_mon, t.tm_year
39         )
40     )
41     print("The time is {}:02:02".format(t.tm_hour, t.tm_min, t.tm_sec))
42     time.sleep(1) # wait a second
43

```

The date is 9 Thursday 9/3/2023
 The time is 15:00:11
 The date is Thursday 9/3/2023
 The time is 15:00:12
 The date is Thursday 9/3/2023
 The time is 15:00:13
 The date is Thursday 9/3/2023
 The time is 15:00:14
 The date is Thursday 9/3/2023
 The time is 15:00:15

Figure 13: RTC result.

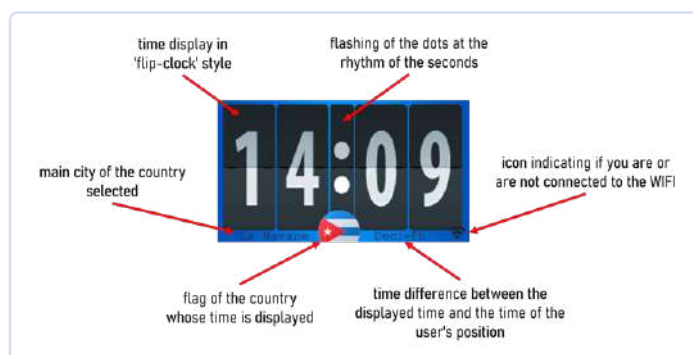


Figure 14:
Final interface.



Figure 15: Digit sprite sheet.

be tedious, but this tutorial clarifies all the necessary basics [18].

What is different from the previous listings?

- > The real-time clock (RTC) management code has been mixed with the code that allows to fetch the time through the NTP server.
- > The time is now displayed, thanks to bitmap images previously prepared in dedicated code. We have to use the `adafruit_imageload` library to load them into memory. To avoid loading an image with each digit change (risk of slowdown), we have to resort to a "sprite sheet," which is also used in animation (see **Figure 15**). Then we have to determine which portion of the image we want to display. The same procedure can apply to the flashing dots.
- > Using the joystick available on the 1.14" LCD Hat for Pico module, we can select a country and visualize the current time in this country as well as the time difference to UTC. To prevent key bounce from the joystick, we use the `adafruit_debouncer` library, which, in turn, needs the `adafruit_ticks` library to work.
- > The flag display also uses a bitmap in the form of a sprite sheet with 25 available countries (see **Figure 16**).



Figure 16: Flag sprite sheet.

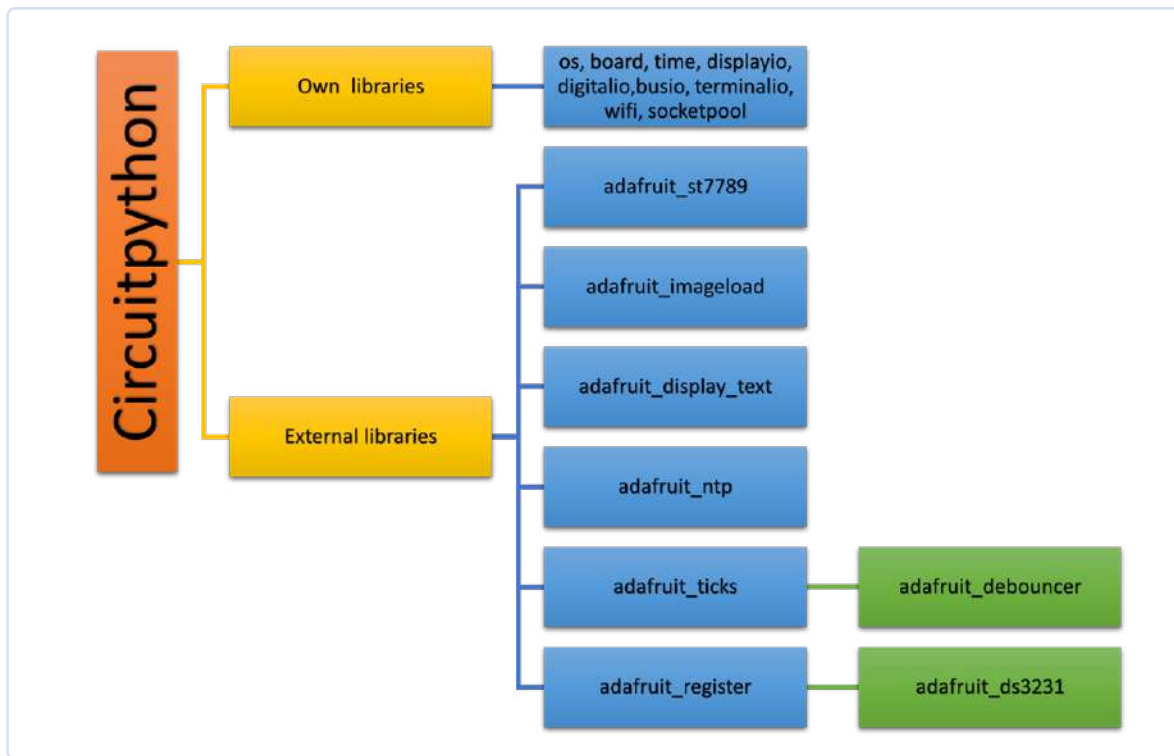


Figure 17: Libraries tree structure.

- › The texts corresponding with the city and the time difference use labels just like we did before when we displayed the time on the LCD.

The tree structure of the internal and external libraries used in this project is shown in **Figure 17**.

Conclusion

There are, of course, many ways to improve this project, such as using a higher-resolution LCD (without exceeding 320×240 though, as you are limited by the power of the Raspberry Pi Pico W), adding audible user alarms or displaying current weather data for your location... These enhancements can be easily coded in CircuitPython thanks to the resources made available by online libraries, or to the many tutorials that can be found on the Adafruit website.

And, why not try to program your own CircuitPython project and appreciate how easy it is to use this language?

- › Download: CircuitPython for the Raspberry Pi Pico W [18]
- › Download: External libraries [19]
- › Documentation: CircuitPython native libraries [20]
- › Documentation: external libraries [21]
- › CircuitPython Essentials tutorials [22], [23]

You can also find other resources in a (French) book I wrote using CircuitPython version 5.3 (see **Figure 18**). It was published by Elektor in 2020, so, obviously, there have been many innovations since that edition, but the language basics remain the same and the code can be easily ported to other modules such as the Raspberry Pi Pico W (which is one of the main strong points of CircuitPython and its 'unified hardware API!'). If you are stuck, you can always send your questions to my email address! ◀

Translated to English by Fanny Maunier — 220633-01



Figure 18: "Initiation au langage circuitpython et à la puce nRF52840" book.

Questions or Comments?

If you have any questions or simply want to propose new article ideas, you can contact the author through email at michael.bottin@univ-rennes.fr, or the Elektor editorial team by email at editor@elektor.com.



Related Products

- › **Raspberry Pi Pico RP2040 W**
<https://elektor.com/20224>
- › **Michael Bottin, Initiation au langage CircuitPython et à la puce nRF52840, Book (French)**
<https://elektor.com/19523>

WEB LINKS

- [1] MicroPython: <https://en.wikipedia.org/wiki/MicroPython>
- [2] CircuitPython: <https://en.wikipedia.org/wiki/CircuitPython>
- [3] Raspberry Pico W from Elektor: <https://elektor.com/raspberry-pi-pico-rp2040-w>
- [4] LCD Hat for Pico: <https://shop.sb-components.co.uk/products/1-14-lcd-hat-for-pico>
- [5] Pico RTC Hat: <https://shop.sb-components.co.uk/products/pico-rtc-hat>
- [6] Download CircuitPython: <https://circuitpython.org/>
- [7] Mu Editor: <https://codewith.mu/en/download>
- [8] Mu Editor Interface: <https://codewith.mu/en/tutorials/1.2/start>
- [9] REPL console: <https://codewith.mu/en/tutorials/1.2/repl>
- [10] Scrolling graph: <https://codewith.mu/en/tutorials/1.2/plotter>
- [11] Keyboard shortcuts: <https://codewith.mu/en/tutorials/1.2/shortcuts>
- [12] TOML file: <https://en.wikipedia.org/wiki/TOML>
- [13] Circup tool: <https://learn.adafruit.com/keep-your-circuitpython-libraries-on-devices-up-to-date-with-circup/>
- [14] UTC (Coordinated Universal Time): <https://timeanddate.com/worldclock/timezone/utc>
- [15] The type of the time value : https://docs.python.org/3/library/time.html#time.struct_time
- [16] adafruit_ds3231 library tutorials and examples: <https://learn.adafruit.com/adafruit-ds3231-precision-rtc-breakout/circuitpython>
- [17] Pico W Pinout: <https://datasheets.raspberrypi.com/picow/PicoW-A4-Pinout.pdf>
- [18] CircuitPython version for the Raspberry Pico W: https://circuitpython.org/board/raspberry_pi_pico_w/
- [19] External libraries: <https://circuitpython.org/libraries>
- [20] CircuitPython native libraries documentation: <https://docs.circuitpython.org/en/latest/shared-bindings/index.html#modules>
- [21] External libraries documentation: <https://docs.circuitpython.org/projects/bundle/en/latest/drivers.html>
- [22] CircuitPython Essentials Tutorials 1: <https://learn.adafruit.com/welcome-to-circuitpython/circuitpython-essentials>
- [23] CircuitPython Essentials Tutorials 2: <https://learn.adafruit.com/welcome-to-circuitpython/>
- [24] Software Download: <https://elektormagazine.com/220633-01>

Genius

The only autonomous wheel in the world.
Innovation that transforms anything into an autonomous mobile robot with simple set up.

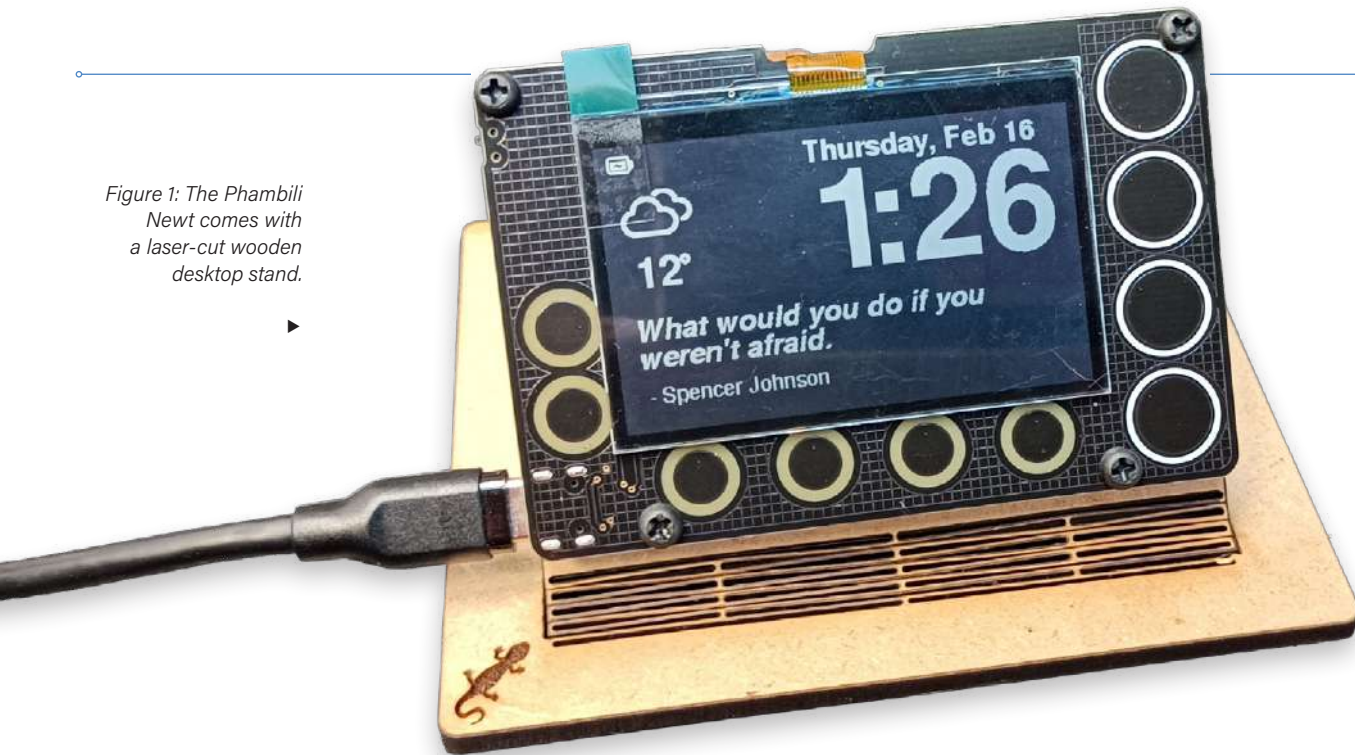


wheel_me

-  Smart navigation
-  State of the art sensor technology
-  Flexibility to adjust your payload
-  Omnidirectional movement
-  Remote access via App
-  In-process charging

more details at www.wheel.me

Figure 1: The Phambili Newt comes with a laser-cut wooden desktop stand.



Build a Cool IoT Display

With the Phambili Newt

By Clemens Valens (Elektor)

Dive into an exploration of the Phambili Newt, a compact and customizable display module that offers more than meets the eye. Let's take a look at its unique features, from basic functionality to the exciting potential of user-programmable applications, making it a promising tool for those interested in the realm of IoT devices.

The Phambili Newt is a battery-powered, always-on, wall-mountable display that can go online to retrieve information and display it. The module is slightly bigger than a credit card and features 10 touchpads together with a 2.7" 240×400-pixel E-ink-like display. Behind the display sits an ESP32-S2 microcontroller that you can program with Arduino, CircuitPython, MicroPython [1], or ESP-IDF. Even though the display is intended for always-on applications, it does have a tiny power slide switch, so it can also be off.

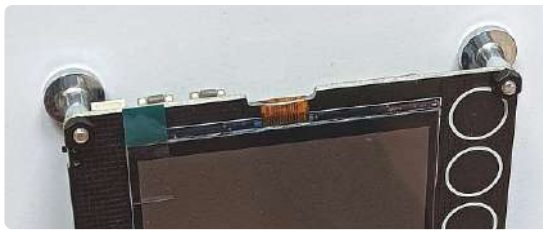
A battery is an option to power the Newt, but one is not included in the kit. Luckily, you can also run it from a USB-C-type phone charger. According to the documentation, a 500 mAh (minimum capacity) Li-Po battery would allow the device to operate for up to two months between charges. The battery must have a so-called 2-way JST connector.

Besides the Newt module, the kit also includes a laser-cut wooden stand and some mounting material. The stand lets you place the device on, for instance, your desk, but it is a bit wobbly when touching the key pads. A more stable option is to mount the four magnetic feet and stick the Newt on a fridge or another metal surface.

What Does the Phambili Newt Do?

Out of the box, the Newt doesn't do a whole lot, as it expects you to configure a Wi-Fi connection first. After switching the module on, it gives instructions on how to connect it to your network. Doing this is easy enough, but a bit slow, and I observed several device reboots before it connected to my network. When it finally did, it displayed the date and time and then got stuck.

Figure 2: With its magnetic feet, you can stick the Newt on a fridge or other metal object. ►



I noticed that the version of the firmware loaded on my Newt was v0.0.11, so I checked for a more recent version. After loading the latest version [2] (v1.1.15) the Newt worked as expected. Upgrading the firmware is easy: Connect the Newt to a computer and copy the new firmware file to the external disk that gets created.

With the proper firmware, the Newt connects quickly to my network and then shows time, date and weather information for my location. The weather data display alternates every three minutes with a quote.

Touching the upper-right key opens a menu at the bottom of the screen. There are three “pages” with alarm and timer, weather and air quality things, a calendar, and other stuff that you might find useful or fun. The menu also gives you access to the settings and firmware upgrade. Unfortunately, the mounting screw is a bit in the way of the menu button (as does the screw close to the lower-left button).

Write Your Own Applications for the Phambili Newt

Even though the base functionality of the Newt is nice, it is probably not the reason why you would want one. Its real power lies in the fact that it's hackable, so to speak. The source code of the firmware is available on GitHub, together with instructions on how to set up the Arduino IDE [3] for writing your own Newt applications.

An I²C connector in Qwiic (SparkFun) format lets you connect sensors and other extensions to the Newt, turning it into a real IoT device instead of just a connected clock.

Conclusion

The ESP32-S2 is a powerful microcontroller, even though it does not feature Bluetooth, but only Wi-Fi. The black-and-white display looks very nice and is fast, unlike normal E-ink displays. Display drawing is instantaneous. The combination of the two makes for a cool module with many application possibilities, especially in the low-power IoT domain. The I²C extension connector provides even more options. ◀

230345-01

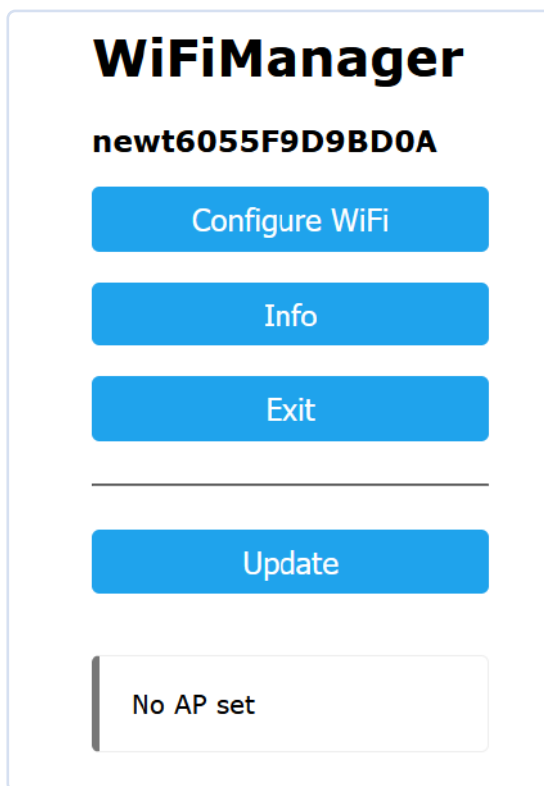


Figure 3: Your destination after switching on the Newt for the first time. ◀

Questions or Comments?

Do you have technical questions or comments about this article? Email clemens.valens@elektor.com to get in touch with the author.



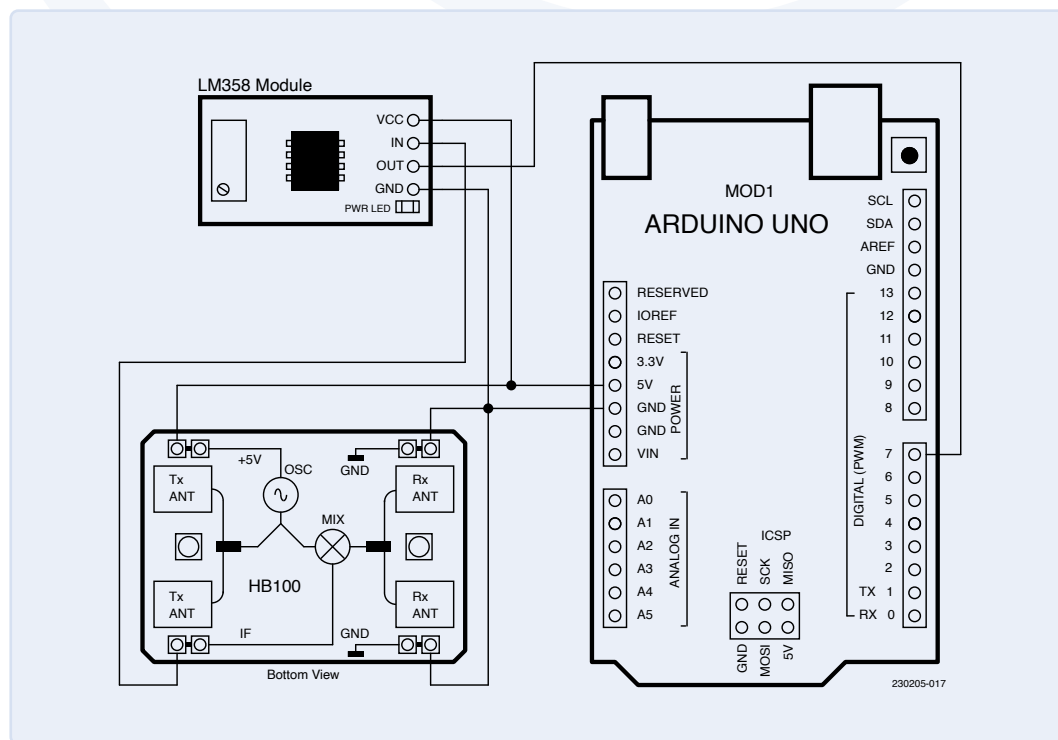
Related Products

- **Phambili Newt 2.7" IoT Display (powered by ESP32-S2)**
<https://elektor.com/20230>
- **SparkFun Environmental Combo Breakout - CCS811/BME280 (Qwiic)**
<https://elektor.com/19580>

WEB LINKS

- [1] Günter Spanner, “MicroPython for the ESP32 and Friends (Part 1)”: <https://elektormagazine.com/articles/micropython-esp32-microcontroller>
- [2] Latest version firmware on Newt : https://phambili-pub.s3.amazonaws.com/Newt.ino_latest.bin
- [3] How to set up the Arduino IDE: https://github.com/Phambili-Tech/Newt_Display/wiki/Arduino-Setup

Theory and Practice



By Stefano Lovati (Italy)

Detecting the movement of humans, other animals, or various objects requires the use of suitable motion sensors. This article delves into a specific type of sensor, valued for its unique capabilities and educational significance: the microwave sensor. This motion-detecting device operates on the Doppler effect, the same principle employed in modern radar systems.

Unlike infrared motion detectors, microwave sensors detect the motion of an object by analyzing microwaves reflected by the object. Compared with other types of sensors, the categories of objects that can be detected are not limited to the human body. Furthermore, the microwave sensor is not affected by ambient temperature, has a wide measurement range and high sensitivity. Because of its properties, it is widely used in industrial devices, transportation, automatic door control, parking sensors, and as a speed meter. Because of its ability to detect different types of objects, in many real-life applications this sensor is combined with another type of sensor to perform targeted and fail-proof detection. For example, the microwave sensor can be paired with an infrared presence sensor (PIR) to more effectively determine the transit or presence of a person, eliminating possible sources of disturbance.



Notes on the Doppler effect

The Doppler effect refers to the change in frequency, or pitch, that occurs when a sound source approaches or moves away from the listener, or vice-versa, when the listener approaches or moves away from the sound source. Any of us have been able to experience this effect on a practical level, noticing how the sound produced by a siren (which for simplicity we assume to be monotonic and with a fixed frequency) is modified when the source approaches or moves away from us. This principle, discovered by Austrian physicist Christian Doppler, not only applies to sound waves, but is also applicable to radio waves, or electromagnetic radiation. This apparent change in frequency between the source of a wave and the receiver is determined precisely by the relative motion between the source and the receiver.

To better understand the Doppler effect, let us assume that the frequency and wavelength of a sound from a source remain constant. When both the source and receiver are stationary, the receiver will hear the same sound frequency produced by the source. This is because the receiver detects the same number of waves per second produced by the source. If, on the other hand, the source, the receiver, or both, are moving toward each other, the receiver will perceive a higher-frequency sound. This is because the receiver will detect more sound waves per second and interpret this as a higher-frequency sound. Conversely, if the source and receiver are moving away from each other, the receiver will detect fewer sound waves per second and perceive a lower-frequency sound.

A graphical representation of the concept just expressed is provided in **Figure 1**. When the vehicle approaches the source (the transmitter), the reflected wave undergoes an increase in frequency; however, when the vehicle moves away, the reflected wave undergoes a decrease in frequency from the original wave.

Some Equations

For a radar system, the value in Hertz of the frequency shift f_D , called the Doppler frequency, can be calculated using the formula

$$f_D = \frac{2 \cdot v}{\lambda} \quad \text{Equation 1}$$

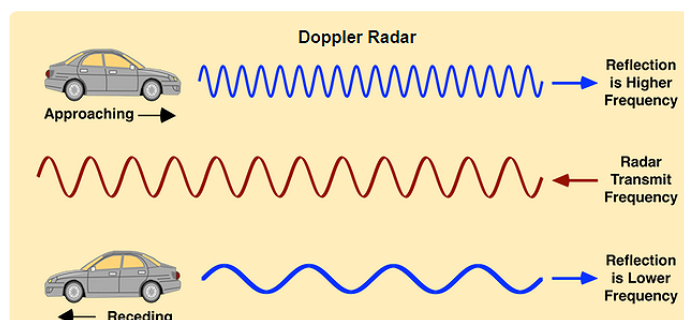


Figure 1: Example of the application of the Doppler effect. (Source: physicsopenlab.org, CC BY 4.0)

where v is the speed of the detected object in m/s and λ is the wavelength in meters. This formula is valid while the velocity v is uniquely radial, thus having no transverse components, as is the case with lateral motion of the object to be detected (the one reflecting the signal) and the source (the transmitter). In the most general case, the above equation takes the form

$$f_D = \frac{2 \cdot v}{\lambda} \cdot \cos \alpha \quad \text{Equation 2}$$

where α is the angle formed by the direction of the transmitted/reflected signal with the direction of motion of the object to be detected. Note how, in the case where the object to be detected moves only perpendicular to the direction of the transmitted signal, the Doppler effect cancels out ($f_D=0$). This explains why in *Top Gun*-style aerial combat, where sophisticated radars are used that exploit precisely the Doppler effect, pilots under radar threat always tend to cross their trajectory perpendicularly with that of the enemy, making radar detection and tracking of their position and speed more complicated.

In the specific case of a radar transmitting electromagnetic waves (such as the one in the sensor used for this article), the formula expressing the modulus, or absolute value, of the Doppler frequency f_D is

$$|f_D| = \frac{2 \cdot v_r}{\lambda} = \frac{2 \cdot v_r \cdot f_{TX}}{c_0} \quad \text{Equation 3}$$

where v_r is the radial velocity, f_{TX} the frequency of the transmitted signal, and c_0 the speed of light.

The HB100 Module

In this article, we will analyze the properties of one of the cheapest Doppler effect-based presence detection sensors: the HB100. It is basically an integrated module that includes an X-band microwave transmitter, a receiver circuit, a mixer, and the whole RF section, realized on a PCB of extremely compact size. In order to reassure readers, let me note that although the frequencies involved are quite high (around 10 GHz), there are no risks for our health, since the transmission power is very low. The operating principle of this module, which is readily available from major electronic component distributors for a few euro, is not only very interesting but also important from an educational point of view. Indeed, the superheterodyne circuit used by the module forms the basis of modern radar systems and is a classic scheme found in many radio frequency receivers. **Figure 2** shows the outward appearance of the module, which is very compact, with the metal casing that acts as a protective shield for the RF part in evidence. In contrast, **Figure 3** shows the rear of the module, which basically acts as a transceiver antenna.

The HB100 miniature motion sensor is a Doppler transceiver module that operates in the X-band at a frequency of 10.525 GHz. Inside it is a dielectric resonator oscillator (DRO) and a pair of antennas, which are

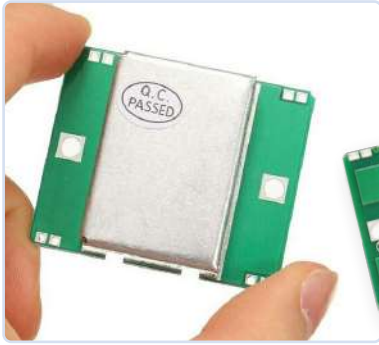


Figure 2: Top view of the HB100 module. (Source: [6])

etched directly on the PCB. This module is an ideal solution for reducing the incidence of false alarms in intrusion detection systems, especially when combined with a conventional passive infrared (PIR) sensor. The sensor can also be used in automatic door-opening systems and vehicle speed meters. The main advantages offered by the HB100 module are as follows:

- Non-contact motion detection;
- Measurement not affected by temperature, humidity, noise, air, dust; the sensor is also suitable for particularly harsh environmental conditions;
- Excellent immunity to radio interference;
- Low output power. The device poses no danger to humans and complies with Federal Communication Commissions (FCC) regulations;
- High detection distance (up to 20 meters);
- Ability to detect the movement of various objects, not just humans;
- High directionality of radio waves;
- Low-power consumption;
- Ability to operate in both CW (Continuous Wave, i.e., continuous transmission of the radio signal) and Pulse (transmission of short pulses that follow one another over time) modes;
- Compact size and very thin.

In **Figure 4**, we can observe the block diagram of the module, which we will now analyze in detail. Let's start with the oscillator which, as previously mentioned, is a DRO, tuned to the frequency of 10.525 GHz. A ceramic disk, usually made of barium titanate ($\text{Ba}_2\text{Ti}_9\text{O}_{20}$) acts as a resonance chamber for the RF energy. The resonant frequency depends on the size and shape of the material. This is exactly the

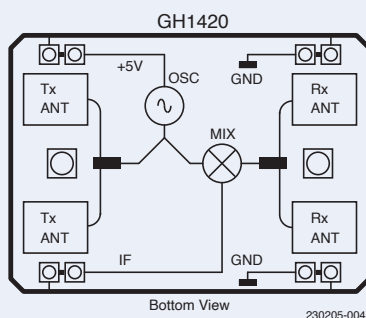


Figure 4: Block diagram of the sensor. (Source: [5])

Figure 3: Bottom view of the HB100 module. (Source: [7])

frequency value of the signal transmitted from the module through the two rectangular antennas (etched on the PCB) visible on the left side of Figure 4. Any obstacle that gets in the way of the transmitted beam will produce a reflected wave, the frequency of which varies according to the movement of that object relative to the transmitting module. In any case, the reflected wave will have a frequency that differs a little from the transmitted one. Handling an RF signal at these frequencies is a daunting task for anyone, both at hardware level and, most importantly, at software level. The solution to this problem lies precisely in the superheterodyne circuit and, precisely, in the RF mixer that we see in Figure 4. Its function is to "combine" the transmitted signal with the one received through the two rectangular antennas, visible on the right in Figure 4. In general, a mixer receives two input signals and produces two output signals that are, respectively, the sum and the difference of the two input signals. Therefore, in our case, two signals would be generated: one with a frequency in the order of 20 GHz (sum of the transmitted signal with the received one) and the other with a frequency of a few Hertz (difference between the transmitted and received signals). This mechanism elegantly solves our problem, as it shifts the problem of measuring an X-band signal (tens of GHz) into the measurement of a low-frequency signal (up to a few hundred Hz, dependent on the speed of the moving object), easily handled by a common low-cost microcontroller such as that on an Arduino.

The signal denoted as IF (Intermediate Frequency) output from the mixer is exactly the difference signal. Note how this process, common to many RF circuits, is also called "down-conversion:" its purpose is, in fact, to down-convert a high frequency so as to simplify its handling. The frequency thus obtained is called the "intermediate frequency" to distinguish it from the original, or "baseband" frequency.

Still looking at Figure 4, we may also note the presence of 4 pins only, which are necessary for connecting the module to a microcontroller or measurement circuit: 2 ground pins, 1 pin for the positive 5 V supply, and 1 pin for the low-frequency IF output signal. In essence, this is to handle a signal with a frequency of only a few Hertz, leaving all the "dirty job" of handling the microwave signal to the built-in hardware of the HB100 module. Actually, there is still a problem, related to the rather low amplitude of the signal produced at the sensor output; we will see later how to deal with this issue.

For practical use, the module should be mounted with the antennas (see Figure 3) facing the area you want to cover with the radiated output, orienting them in such a way as to obtain the best coverage. The antenna radiation diagrams, both in azimuth and elevation, can be seen in **Figure 5**.

Output Signal

The IF pin represents the output signal of the module, corresponding to the frequency shift determined by the magnitude of the movement of any object illuminated by the antenna. The amplitude of this frequency shift, affected by the Doppler effect, is proportional to the reflection of

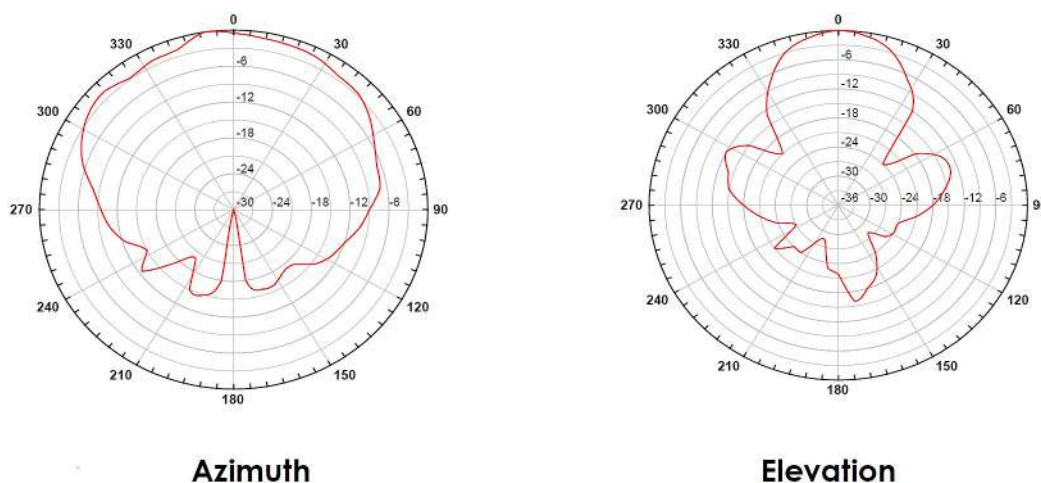


Figure 5: Radiation diagrams of the antenna. (Source: [1])

the transmitted energy and is in the order of a few microvolts (μV). For this reason, a low-frequency, high-gain amplifier is normally connected to the IF pin to allow amplification of the output signal. More precisely, the frequency of the Doppler shift is proportional to the velocity of the motion. As an example, keep in mind that a typical human walk produces a frequency shift of less than 100 Hz. The received signal strength (RSS) corresponds to the voltage measured at the IF output pin. The Doppler frequency can be calculated using the equations earlier. In the case where the subject illuminated by the sensor moves only along the radial line, either approaching or receding, the previous formula is simplified and takes the following form where v_r is the radial velocity, expressed in km/h:

$$f_D = 19.49 \cdot v_r \quad \text{Equation 4}$$

The technical specifications of the module can be summarized as follows:

- > Operating voltage: $5\text{ V} \pm 0.25\text{ V}$;
- > Operating current (in continuous transmission mode): 60 mA maximum, 37 mA typical;
- > Size: 61.2 x 61.2 mm;
- > Sensing distance: between 2 m and 16 m;
- > Transmission frequency: 10.525 GHz;
- > Frequency accuracy: 3 MHz;
- > Minimum output power: 13 dBm EIRP;
- > Harmonic emission: $< -10\text{ dBm}$;
- > Average current: 2 mA typical;
- > Minimum transmission pulse width: 5 μs ;
- > Minimum duty cycle: 1 %

Signal-Conditioning Circuit

The output of the HB100 sensor, available on the IF pin, is a low-frequency signal corresponding to the frequency shift affected by the received signal. The problem is that only an unamplified signal

of very small amplitude (a few μV) is available on this output and, therefore, in practical applications, requires proper amplification, preferably accompanied by a low-pass filter to eliminate any spurious frequencies above a few hundred Hertz. Both the datasheet [1] and the application note [2] of the sensor are very useful in this regard. The latter, in particular, shows two possible diagrams related to the signal-conditioning circuit. The first, visible in **Figure 6**, is applicable to the continuous transmission (CW) mode and is based on the LM324 high-gain operational amplifier manufactured by Texas Instruments.

The second circuit, visible in **Figure 7**, is to be used in the Pulse mode, in which the transmitter emits a sequence of pulses at a given

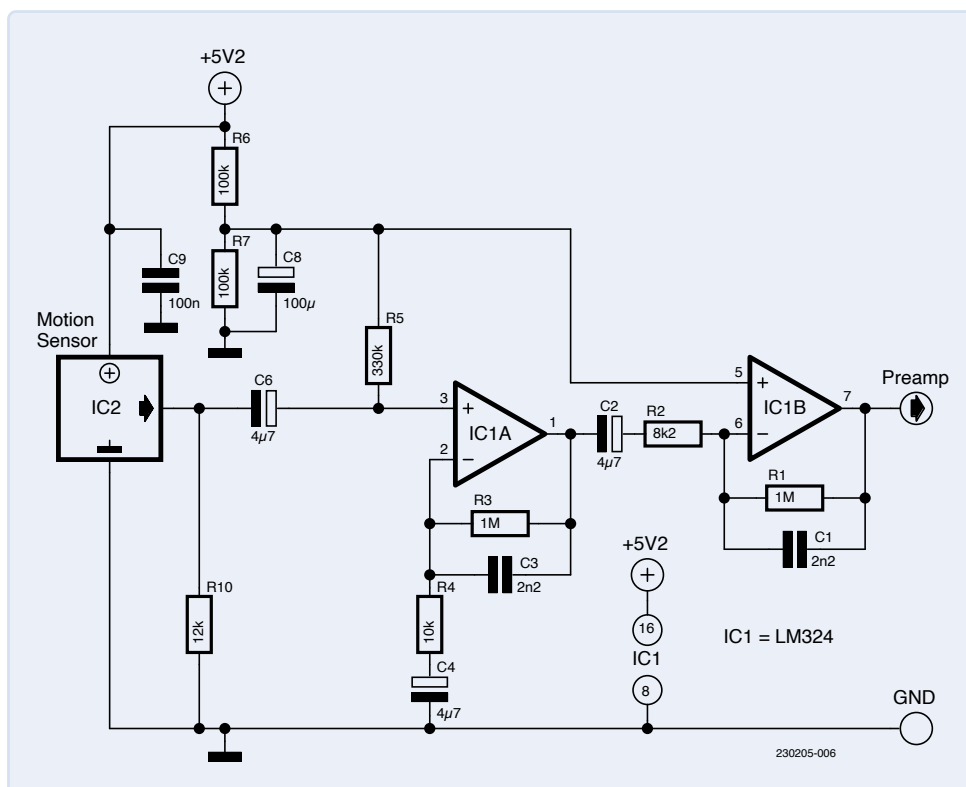


Figure 6: Conditioning circuit for CW mode. (Source: [5])

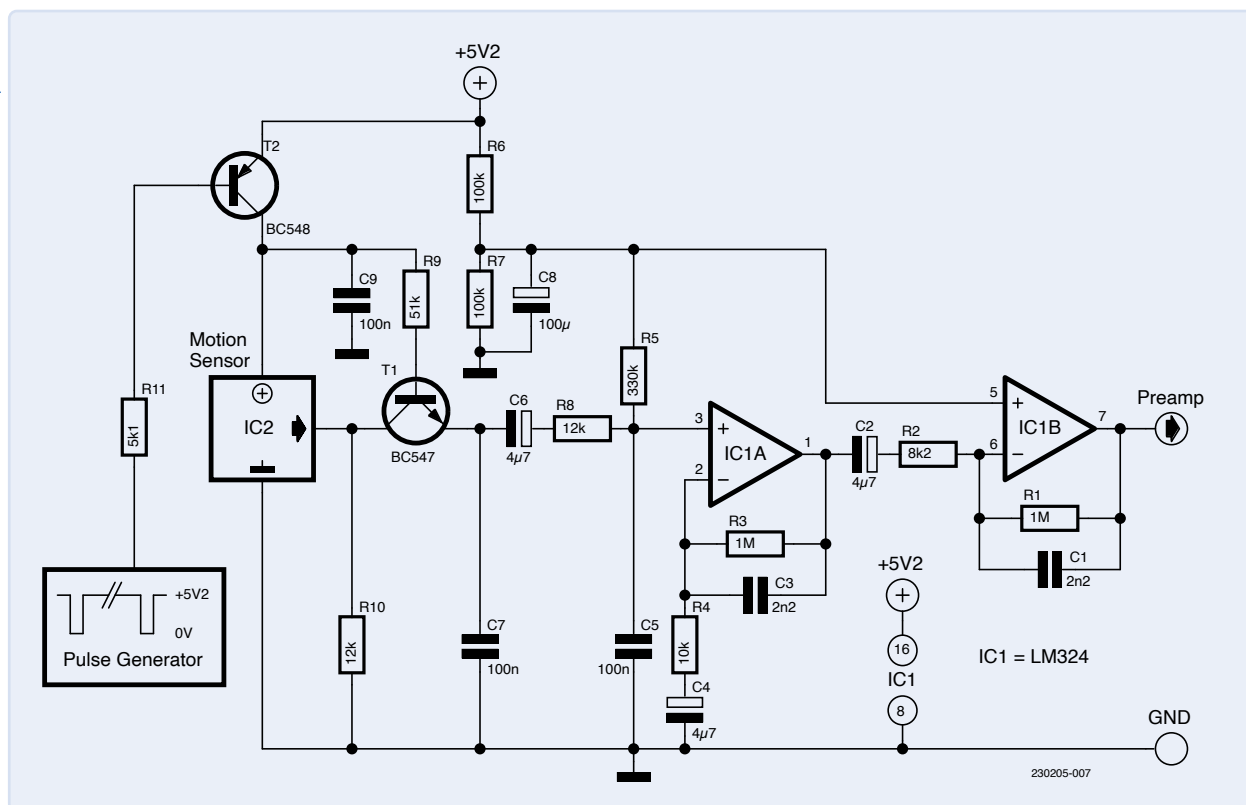


Figure 7: Conditioning circuit for Pulse mode. (Source: [5])

frequency (known as PRF, short for Pulse Repetition Frequency) and with a given duty cycle. Specifically, the HB100 module manufacturer suggests using a $PRF = 2 \text{ kHz}$ and duty cycle = 4%.

This circuit, in addition to the LM324 high-gain operational amplifier, employs two transistors. As visible in the lower-left corner of Figure 7, the pulse mode can be activated by directly acting on the module supply voltage, in such a way to respect the PRF and duty cycle defined (and recommended) by the manufacturer. In other words, in Pulse mode, the added series switching transistor sets the ON condition every 0.5 ms (2 kHz) with a duty cycle of 4% (4% ON).

However, to simplify tests using the sensor, a general-purpose operational amplifier, already mounted on PCB with pin-header and potentiometer for gain adjustment, was chosen. The module, which is very inexpensive and readily available, is based on the LM358, as shown in Figure 8. The LM358 provides single-channel amplification with variable gain between 1x and 100x, which can be adjusted using the potentiometer provided: turning the screw clockwise decreases the gain; turning the screw counter-clockwise increases the gain. Furthermore, the module includes an LED to indicate the correct power supply for it. The LM358 op-amp can handle signals with frequencies up to 700 kHz (so well above the band we are interested in) and can be powered with a voltage between 3 V and 32 V. Figure 8 shows the module's interface pins: V_{CC} , GND, Signal In and Signal Out.

Testing with Arduino

To test the HB100 Doppler sensor, we can use a common Arduino UNO board and the LM358 amplifier module or, alternatively, any other high-gain, low-noise, single-channel, single-supply amplifier. A special sketch will be run on an Arduino UNO board that can detect the frequency of the signal output from the module, and possibly

derive from this the speed at which the potential "target" is moving. The sketch in question uses a library specially developed for accurate frequency measurement. Indeed, for the measurement of frequencies in the audio band or lower, the period length of the input signal must be determined very accurately. For this purpose, the library uses a highly accurate "Counter and Capture" module available in the hardware architecture of the ATmega microcontroller. The library returns the period duration measurement, expressed as an integer with 1/16 μs resolution. To derive the frequency, simply divide the clock frequency by the value returned by the library. In our case, the clock frequency will be set to 16,000,400 to compensate for any inaccuracies on the Arduino board. The library is not currently included in the official Arduino library,

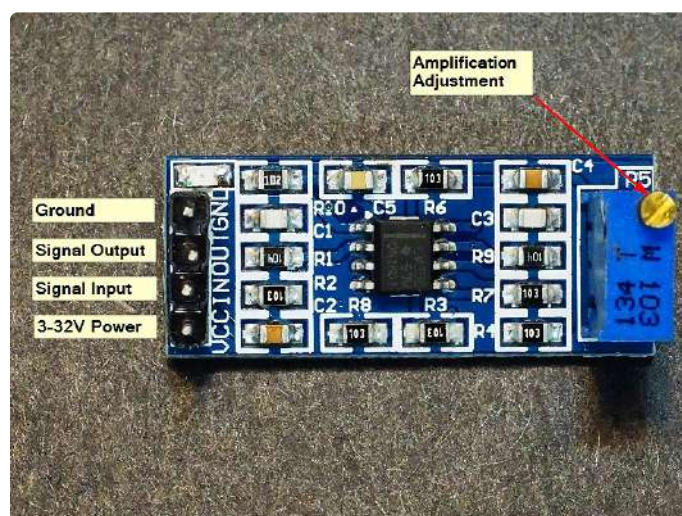


Figure 8: The LM358 amplifier module.

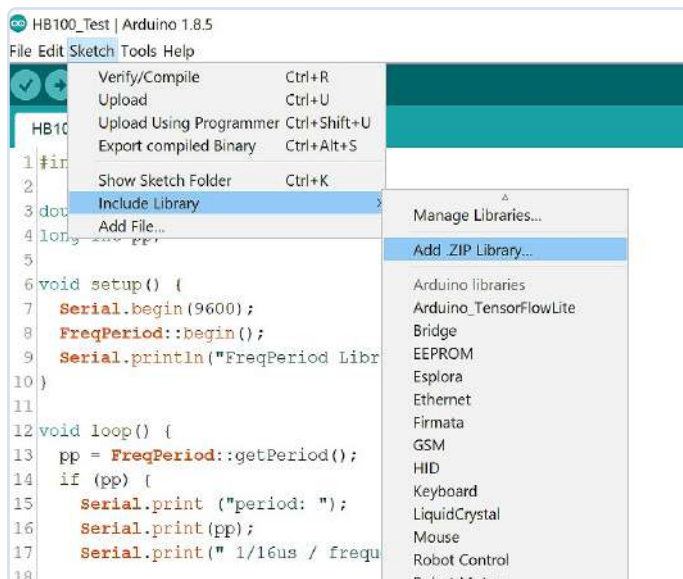


Figure 9: Installation of the FreqPeriod library.

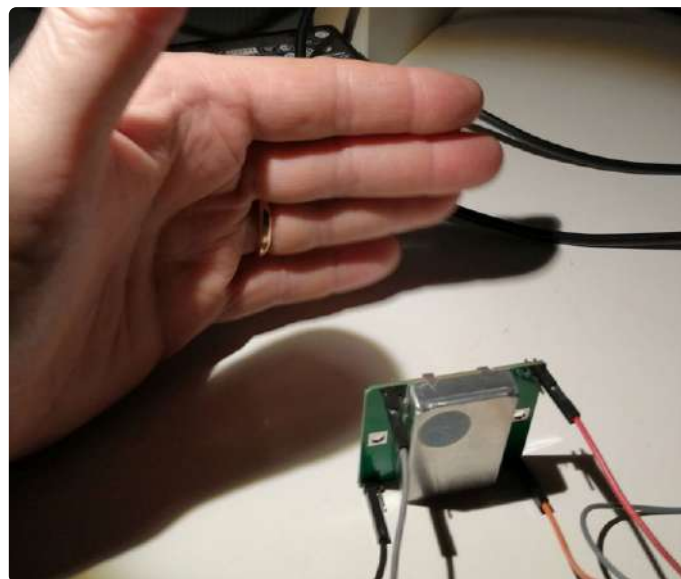


Figure 10: Rapid hand movements in front of the sensor antenna.

but it can be easily installed in the IDE after downloading the .zip file available at [3]. To do this, open the Arduino IDE, select the menu item *Sketch Include Library Add .ZIP Library...*, as shown in **Figure 9**. Then, select the .zip file of the previously downloaded library. Close and restart the IDE to make the new library available in the Arduino environment.

The sketch can be seen in **Listing 1**. It is very compact and relatively simple. In the `setup()` function, the serial line (at 9600 bps), and the `FreqPeriod` library are initialized via the `begin()` method. In the `loop()` function, the period value of the input signal is first acquired via the call to the `getPeriod()` method, repetitively. Subsequently, the corresponding values of frequency (in Hertz) and radial speed (in km/h) are calculated and displayed.

In order to verify the operation of the sketch, we must wire the HB100 sensor and the LM358 amplifier module to the Arduino board. The connections can be seen in the **main picture** at the beginning of this article.

We then turn on the circuit and activate Arduino IDE's Serial Monitor to observe the logs produced. By moving an object, for example our hand (see **Figure 10**), in front of the HB100 sensor antenna (which we remember is located on the back of the PCB, on the opposite side from the metal container), we should see a change in the speed calculated by the sketch.

An example of output sent via the serial interface can be seen in **Figure 11**.

Using this speed value, one can then construct even very complex algorithms with applications such as door operation (including sliding doors), automatic staircase light switching, pedestrian gate



Listing 1: Arduino sketch.

```
#include <FreqPeriod.h>
double lfrq;
long int pp;

void setup() {

    Serial.begin(9600);
    FreqPeriod::begin();
    Serial.println("FreqPeriod Library Test");

}

void loop() {

    pp = FreqPeriod::getPeriod();

    if (pp) {
        Serial.print("period: ");
        Serial.print(pp);
        Serial.print(" 1/16us / frequency: ");
        lfrq = 16000400.0 / pp;
        Serial.print(lfrq);
        Serial.print(" Hz ");
        Serial.print(lfrq/19.49);
        Serial.println(" km/h ");
    }

}
```

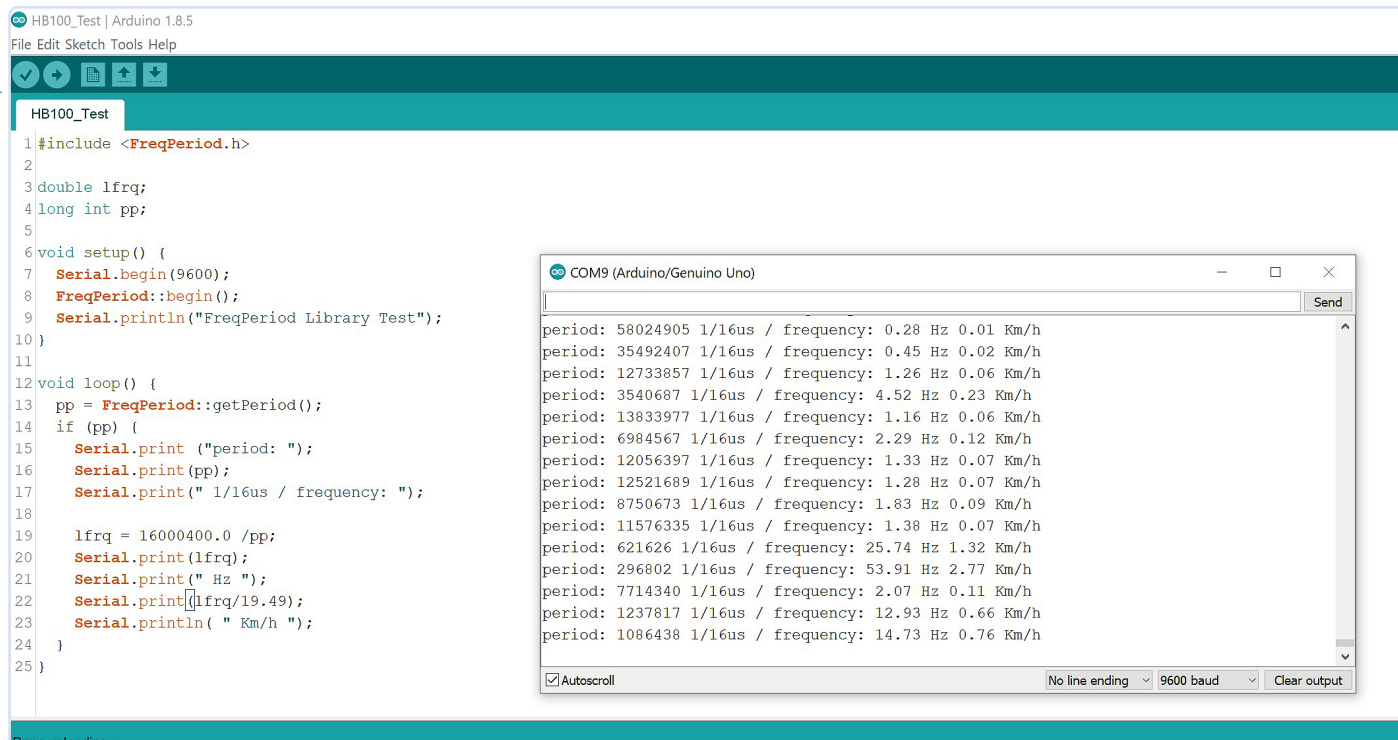



Figure 11: Output in Serial Monitor.

opening, intrusion detection systems, video surveillance, and more. The sensitivity of the module itself is good enough, provided it operates within its maximum range, which is about 20 metres. Performance can be improved by using an amplifier circuit with higher gain than the one (100×) offered by the LM358.

Conclusion

The aim of this article was to present a very interesting and not very well-known sensor, the HB100 motion Doppler sensor. Particularly inexpensive and easily interfaced with a microcontroller, the HB100 offers the ability to learn and experiment with important signal-detection concepts and techniques that underlie sophisticated and expensive commercial radars. There is no shortage of applications for the sensor: No doubt many makers are already thinking about further, future developments of it. ◀

230205-01

Questions or Comments?

If you have technical questions, feel free to e-mail the Elektor editorial team at editor@elektor.com.

About the Author

After graduating in Electronic Engineering at Politecnico di Milano, Stefano started his career as a firmware and software developer. Over the years, he has experienced and 'ridden' the progressive transformation of the embedded world, from the first 8-bit micro-processors - programmable only in assembler- to the most recent Soc, FPGA, DSP and programmable logic with outstanding performance and features. He has an all-round interest in technology and electronics, dedicating part of his free time to the study of new components and the realization of small projects.



Related Products

- > **YDLIDAR X2 Lidar – 360-degree Laser Range Scanner (8 m)**
<https://elektor.com/18941>
- > **Arduino Uno Rev3**
<https://elektor.com/15877>
- > **Arduino Uno Mini (Limited Edition)**
<https://elektor.com/20098>

WEB LINKS

- [1] HB100 Module Datasheet: https://limpkin.fr/public/HB100/HB100_Microwave_Sensor_Module_Datasheet.pdf
- [2] HB100 Application Note: https://limpkin.fr/public/HB100/HB100_Microwave_Sensor_Application_Note.pdf
- [3] Frequency measurement library: <https://github.com/Jorge-Mendes/Agro-Shield/tree/master/OtherRequiredLibraries/FreqPeriod>
- [4] Software on this project's web page: <https://elektormagazine.com/230205-01>
- [5] Circuit diagrams: https://mantech.co.za/Datasheets/Products/MSAN-001_AGILSENSE.pdf
- [6] Source of photo (top view): https://mantech.co.za/Datasheets/Products/HB100_RADAR.pdf
- [7] Source of photo (bottom view):
<https://kuongshun-ks.com/uno/uno-sensor/hb100-microwave-doppler-radar-wireless-module-moti.html>

A Bare-Metal Programming Guide (Part 1)

For STM32 and Other Controllers

By Sergey Lyubka (Ireland)

Do you want to program microcontrollers down at their no-frills level, where you can get a deeper understanding of how they actually work? This guide written for developers will help you get started using just the GCC compiler and a reference manual. The fundamentals learned here will help you understand better how frameworks such as Cube, Keil, and Arduino do their job. In this two-part guide, we will use the STM32F429 controller on a Nucleo-F429ZI board, but, what you learn here can easily be adapted to other microcontrollers.

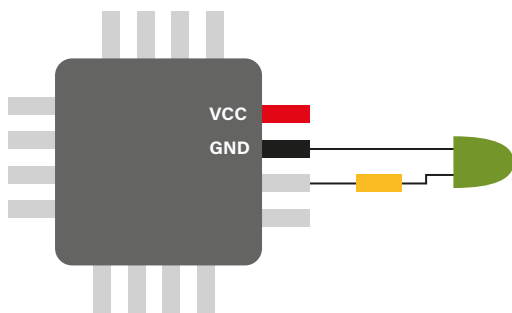


Figure 1: Firmware code can set a high or low voltage on a signal pin, making an LED blink.

A microcontroller (μC , or MCU) is a small computer. Typically, it has a CPU, RAM, flash memory to store firmware code, and a bunch of pins that stick out. Some pins are used to power the MCU, usually marked as GND (ground) and VCC. Other pins are used to communicate with the MCU by means of a high or low voltage applied to those pins. One of the simplest means of communication is an LED attached to a pin: One LED contact is attached to the ground pin (GND), and another contact is attached to a signal pin via a current-limiting resistor. Firmware code can set a high or low voltage on a signal pin, making the LED blink (**Figure 1**).

Memory and Registers

The address space of a 32-bit-MCU, for example the STM32F429 by STMicroelectronics, is divided into regions. For example, one memory region (at a specific address) is mapped to the internal MCU flash. Instructions in firmware are read and executed by reading from that memory region. Another region is RAM, which is also

Required Hardware and Tools

Throughout the guide, we will be using a Nucleo-F429ZI development board (available at Mouser and other distributors). To follow this tutorial, download the STM32F429 MCU reference manual [1] and then the user manual for the development board [2].

To proceed, the following tools are required:

ARM GCC, <https://launchpad.net/gcc-arm-embedded> - for compiling and linking

GNU make, <https://gnu.org/software/make> - for build automation

ST link, <https://github.com/stlink-org/stlink> - for flashing

We show here the setup instructions for Linux (Ubuntu).

Start a terminal, then execute:

```
$ sudo apt -y update
$ sudo apt -y install gcc-arm-none-eabi make
stlink-tools
```

To set the tools up on a Mac or Windows PC, see [3].

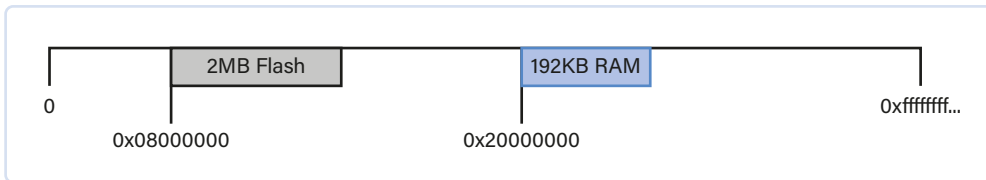


Figure 2: STM32F429 flash and RAM region locations.

mapped to a specific address. We can read and write any values to the RAM region.

From the STM32F429 reference manual [1], we can take a look at section 2.3.1 and see that the RAM region starts at address 0x20000000 and has size of 192 KB. From section 2.4, we learn that flash memory is mapped at address 0x08000000. Our MCU has 2 MB flash, so flash and RAM regions are located as in **Figure 2**.

From the manual, we also learn that there are many more memory regions. Their address ranges are given in the section 2.3, “Memory map.” For example, there is a “GPIOA” region that starts at 0x40020000 and has length of 1 KB.

These memory regions correspond to a different “peripherals” inside the MCU — a piece of silicon circuitry that makes certain pins behave in a special way. A peripheral memory region is a collection of 32-bit registers. Each register is a 4-byte memory range at a certain address, which maps to a certain function of the given peripheral. By writing values into a register — in other words, by writing a 32-bit value at a certain memory address — we can control how a given peripheral should behave. By reading these registers, we can read back a peripheral’s data or configuration.

There are many different peripherals. One of the simpler ones is GPIO (General Purpose Input Output), which allows the user to set MCU pins into “output mode” and set a high or low voltage on them. Or, set pins into an “input mode” and read voltage values from them. There is a UART peripheral that can transmit and receive serial data over two pins using a serial protocol. There are many other peripherals.

Often, there are multiple “instances” of the same peripheral, for example GPIOA, GPIOB, and so on, which control different sets of MCU pins. Likewise, there could be UART1, UART2, etc., which enable the implementation of multiple UART channels. On the STM32F429, there are several GPIO and UART peripherals.

For example, the GPIOA peripheral starts at 0x40020000, and we can find the GPIO register description in section 8.4 [1]. The reference manual says that `GPIOA_MODER` register has offset 0, which means that its address is `0x40020000 + 0`. In **Figure 3**, we can see the format of the register.

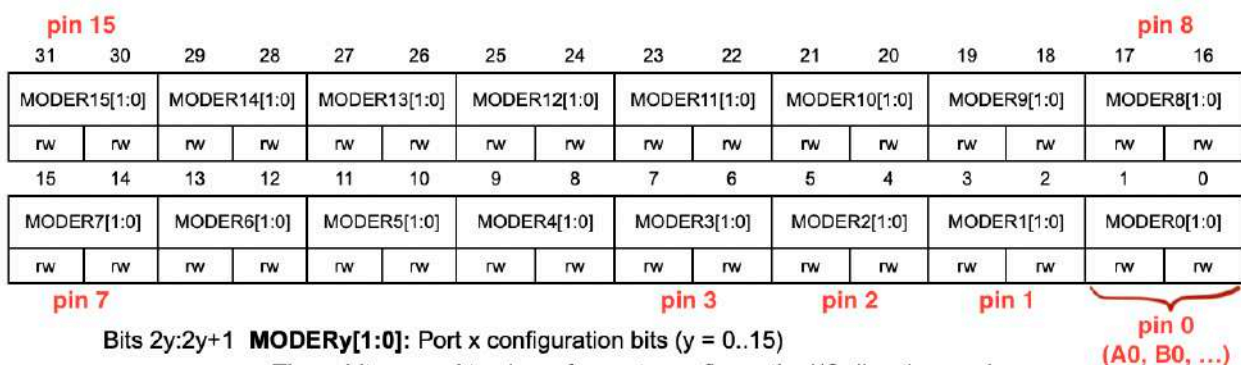
The manual shows that the 32-bit MODER register is a collection of 2-bit values; 16 in total. Therefore, one MODER register controls 16 physical pins, Bits 0...1 control pin 0, bits 2...3 control pin 1, and so on.

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports



Bits $2y:2y+1$ **MODERy[1:0]**: Port x configuration bits ($y = 0..15$)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

Figure 3: We can find the GPIO register description in the reference manual. One MODER register controls 16 physical pins. (Source: [1])

The 2-bit value encodes the pin mode: 0 means input, 1 means output, 2 means “alternate function” — some specific behavior described elsewhere, and 3 means analog. Since the peripheral name is *GPIOA*, pins are named “A0,” “A1,” etc. For peripheral *GPIOB*, pin naming would be “B0,” “B1...”

If we write 32-bit value 0 to the *MODER* register, we’ll set all 16 pins, from A0 to A15, to input mode:

```
* (volatile uint32_t *) (0x40020000 + 0) = 0;
// Set A0...A15 to input mode
```

Note the `volatile` qualifier. Its meaning will be covered later. By setting individual bits, we can selectively set specific pins to a desired mode. For example, this snippet sets pin A3 to output mode:

```
* (volatile uint32_t *) (0x40020000 + 0) &= ~(3 << 6);
// Clear bit range 6...7
* (volatile uint32_t *) (0x40020000 + 0) |= 1 << 6;
// Set bit range 6...7 to 1
```

Let me explain those bit operations. Our goal is to set bits 6...7, which are responsible for pin 3 of the *GPIOA* peripheral, to a specific value (1, in our case). This is done in two steps. First, we must clear the current value of bits 6...7, because they may hold some value already. Then we must set the relevant bits of 6...7 to get the value we want.

So, first, we must set bit range 6...7 (two bits at position 6) to 0. How do we set a number of bits to 0? You can see the four steps in **Table 1**.

Note that the last operation, logical AND, turns N bits at position X to 0 (because they are ANDed with 0), but retains the value of all other bits (because they are ANDed with 1). Retaining the existing value is important, because we don’t want to change settings in other bit ranges. So, in general, if we want to clear N bits at position X we can write the following:

```
REGISTER &= ~(2^N - 1) << X;
```

Table 1. Setting specific bits to 0.

Action	Expression	Bits (first 12 of 32)
Get a number with N contiguous bits set: 2^N-1 , here $N=2$	3	000000000011
Shift that number X positions left	$(3 << 6)$	000011000000
Invert the number: Turn zeros to ones, and ones to zeroes	$\sim(3 << 6)$	111100111111
Logical AND with existing value	$VAL \&= \sim(3 << 6)$	xxxx00xxxxxx

And, finally, we want to set a given bit range to the value we want. We shift that value X positions left, and OR with the current value of the whole register (in order to retain other bits’ values):

```
REGISTER |= VALUE << X;
```

Human-Readable Peripheral Programming

In the previous section, we learned that we can read and write a peripheral register by directly accessing certain memory addresses. Let’s look at the snippet that sets pin A3 to output mode:

```
* (volatile uint32_t *) (0x40020000 + 0) &= ~(3 << 6);
// Clear bit range 6...7
* (volatile uint32_t *) (0x40020000 + 0) |= 1 << 6;
// Set bit range 6...7 to 1
```

That is pretty cryptic. Without extensive comments, such code would be quite hard to understand. We can rewrite this code to a much more readable form. The idea is to represent the entire peripheral as a structure that contains 32-bit fields. Let’s see what registers exist for the *GPIO* peripheral in section 8.4 of the reference manual. They are *MODER*, *OTYPER*, *OSPEEDR*, *PUPDR*, *IDR*, *ODR*, *BSRR*, *LCKR*, *AFR*. Their offsets are 0, 4, 8, etc. That means we can represent them as a structure with 32-bit fields, and create a `#define` for *GPIOA*:

```
struct gpio {
    volatile uint32_t MODER, OTYPER, OSPEEDR,
                    PUPDR, IDR, ODR, BSRR, LCKR, AFR[2];
};
#define GPIOA ((struct gpio *) 0x40020000)
```

Then, for setting *GPIO* the pin mode, we can define a function:

```
// Enum values are per reference manual: 0, 1, 2, 3
enum {GPIO_MODE_INPUT, GPIO_MODE_OUTPUT,
      GPIO_MODE_AF, GPIO_MODE_ANALOG};

static inline void gpio_set_mode
    (struct gpio *gpio, uint8_t pin, uint8_t mode) {
    gpio->MODER &= ~(3U << (pin * 2));
    // Clear existing setting
    gpio->MODER |= (mode & 3) << (pin * 2);
    // Set new mode
}
```

Now, we can rewrite the snippet for A3 like this:

```
gpio_set_mode(GPIOA, 3 /* pin */, GPIO_MODE_OUTPUT);
// Set A3 to output
```

Our MCU has several *GPIO* peripherals (also called “banks”): A, B, C, ... K. From section 2.3, we see that they are 1 KB apart: *GPIOA* is at address 0x40020000, *GPIOB* is at 0x40020400, and so on:

```
#define GPIO(bank) ((struct gpio *)
```



```
(0x40020000 + 0x400 * (bank)))
```

We can define pin numbering that includes the bank and the pin number. To do that, we use a 2-byte `uint16_t` value, where the upper byte indicates the GPIO bank, and the lower byte the pin number:

```
#define PIN(bank, num) (((bank) - 'A') << 8) | (num)
#define PINNO(pin) (pin & 255)
#define PINBANK(pin) (pin >> 8)
```

This way, we can specify pins for any GPIO bank:

```
uint16_t pin1 = PIN('A', 3); // A3 - GPIOA pin 3
uint16_t pin2 = PIN('G', 11); // G11 - GPIOG pin 11
```

Let's rewrite the `gpio_set_mode()` function to take our pin specification:

```
static inline void gpio_set_mode(uint16_t pin,
uint8_t mode) {
    struct gpio *gpio = GPIO(PINBANK(pin));
    // GPIO bank
    uint8_t n = PINNO(pin); // Pin number
    gpio->MODER &= ~(3U << (n * 2));
    // Clear existing setting
    gpio->MODER |= (mode & 3) << (n * 2);
    // Set new mode
}
```

Now, the code for A3 is self-explanatory:

```
uint16_t pin = PIN('A', 3); // Pin A3
gpio_set_mode(pin, GPIO_MODE_OUTPUT); // Set to output
```

Note that we have created a useful initial API for the GPIO peripheral. Other peripherals, such as UART (serial communication) and others, can be implemented in a similar way. This is good programming practice as it makes code self-explanatory and human-readable.

MCU Boot and Vector Table

When an ARM MCU boots, it reads a so-called “vector table” from the beginning of flash memory. A vector table is a concept common to all ARM MCUs: An array of 32-bit addresses of interrupt handlers. The first 16 entries are reserved by ARM and are common to all ARM MCUs. The rest of interrupt handlers are specific to the given MCU — these are interrupt handlers for peripherals. Simpler MCUs with few peripherals have few interrupt handlers, and more complex MCUs have many.

The vector table for the STM32F429 is documented in Table 62 of the reference manual [1]. From there, we learn that there are 91 peripheral handlers, in addition to the standard 16.

Every entry in the vector table is an address of a function that the MCU executes when a hardware interrupt (IRQ) is triggered.

The exceptions are the first two entries, which play a key role in the MCU boot process. These two first values are an initial stack pointer and the address of the boot function (a firmware entry point to execute).

So, now we know that we must make sure that our firmware should be composed in a way that the second 32-bit value in the flash should contain the address of our boot function. When the MCU boots, it will read that address from flash, and jump to our boot function.

Minimal Firmware

Let's create a file, *main.c*, and specify our boot function, which initially does nothing (falls into infinite loop), and additionally specify a vector table that contains 16 standard entries and 91 STM32 entries. In your editor of choice, create and open the *main.c* file and enter the following:

```
// Startup code
__attribute__((naked, noreturn)) void _reset(void) {
    for (;;) (void) 0; // Infinite loop
}

extern void _estack(void); // Defined in link.ld

// 16 standard and 91 STM32-specific handlers
__attribute__((section(".vectors")))
void (*tab[16 + 91])(void) = {_estack, _reset};
```

For function `_reset()`, we have used GCC-specific attributes `naked` and `noreturn` — they mean that the standard function prologue and epilogue should not be created by the compiler, and that the function does not return.

The `void (*tab[16 + 91])(void)` expression means: define an array of 16 + 91 pointers to functions, that return nothing (`void`), and take two arguments. Each such function is an IRQ handler (Interrupt ReQuest handler). An array of those handlers is called a vector table.

The vector table `tab` we put in a separate section called `.vectors` — we need that later to tell the linker to put that section right at the beginning of the generated firmware, consecutively, at the beginning of flash memory. The first two entries are the value of the stack pointer register and the firmware's entry point. We leave the rest of the vector table filled with zeroes.

Compilation

Let's compile our code. Start a terminal (or a Command Prompt in Windows) and execute:

```
$ arm-none-eabi-gcc -mcpu=cortex-m4 main.c -c
```

That works! The compilation produced a file, *main.o*, which contains our minimal firmware that does nothing. The *main.o* file is in ELF

binary format, which contains several sections. Let's see them in **Listing 1**.

Note that the VMA/LMA addresses for sections are set to 0, meaning that *main.o* is not yet complete firmware, because it does not contain the information about where those sections should be loaded in the address space. We need to use a linker to produce complete the firmware file, *firmware.elf*, from *main.o*.

The `.text` section contains firmware code, which in our case is just a `_reset()` function, two bytes long — a jump instruction to its own address. There is an empty `.data` section and an empty `.bss` section [8] (for uninitialized, but declared variables, this section is generally filled with 0). Our firmware will be copied to the flash region at offset 0x8000000, but our data section should reside in RAM — therefore our `_reset()` function should copy the contents of the `.data` section to RAM. Also, it has to write zeroes to the entire `.bss` section. In our case, the `.data` and `.bss` sections are empty, but let's modify our `_reset()` function anyway, to handle them properly.

To do all that, we must know where the stack starts, and where the data and bss sections start. This we can specify in the *linker script*, which is a file with the instructions to the linker about where to put various sections in the address space, and which symbols to create.

Linker Script

Create a file called `link.ld`, and copy and paste the contents from [4]. Below is the explanation:

```
ENTRY(_reset);
```

This line tells the linker the value of the “entry point” attribute in the generated ELF header — so this is a duplicate of what a vector table has. This is an aid for a debugger (such as Ozone, described in the second installment of this guide) that helps us to set a break-

point at the beginning of the firmware. A debugger does not know about a vector table, so it relies on the ELF header.

```
MEMORY {
flash(rx) : ORIGIN = 0x08000000, LENGTH = 2048k
sram(rwx) : ORIGIN = 0x20000000, LENGTH = 192k /*
remaining 64k in a separate address space */
}
```

This tells the linker that we have two memory regions in the address space, and their addresses and sizes.

```
_estack = ORIGIN(sram) + LENGTH(sram); /* stack points
to end of SRAM */
```

This tells the linker to create a symbol, `_estack`, containing a value at the very end of the RAM region of memory. That will be our initial stack value!

```
.vectors : { KEEP(*(.vectors)) } > flash
.text : { *(.text*) } > flash
.rodata : { *(.rodata*) } > flash
```

These lines tell the linker to put the vector table in flash first, followed by the `.text` section (firmware code), followed then by read-only data `.rodata`.

Next comes the `.data` section:

```
.data : {
_sdata = .; /* .data section start */
*(.first_data)
*(.data SORT(.data.*))
_edata = .; /* .data section end */
} > sram AT > flash
_sidata = LOADADDR(.data);
```



Listing 1: Compilation of main.o

```
$ arm-none-eabi-objdump -h main.o
```

Idx	Name	Size	VMA	LMA	File off	Algn
0	.text	00000002	00000000	00000000	00000034	2**1
	CONTENTS, ALLOC, LOAD, READONLY, CODE					
1	.data	00000000	00000000	00000000	00000036	2**0
	CONTENTS, ALLOC, LOAD, DATA					
2	.bss	00000000	00000000	00000000	00000036	2**0
	ALLOC					
3	.vectors	000001ac	00000000	00000000	00000038	2**2
	CONTENTS, ALLOC, LOAD, RELOC, DATA					
...						

Listing 2: Startup code.

```
int main(void) {
    return 0; // Do nothing so far
}

// Startup code
__attribute__((naked, noreturn)) void _reset(void) {
    // memset .bss to zero, and copy .data section to RAM region
    extern long _sbss, _ebss, _sdata, _edata, _sidata;
    for (long *src = &_amp;_sbss; src < &_amp;_ebss; src++) *src = 0;
    for (long *src = &_amp;_sdata, *dst = &_amp;_sidata; src < &_amp;_edata;) *src++ = *dst++;

    main(); // Call main()
    for (;;) (void) 0; // Infinite loop in the case if main() returns
}
```

Note that we tell the linker to create `_sdata` and `_edata` symbols. We'll use them to copy the data section to RAM in the `_reset()` function.

Same for the `.bss` section:

```
.bss : {
    _sbss = .; /* .bss section start */
    *(.bss SORT(.bss.*) COMMON)
    _ebss = .; /* .bss section end */
} > sram
```

Startup Code

Now we can update our `_reset()` function. We copy the `.data` section to RAM, and initialize the `.bss` section to zeroes. Then, we call the `main()` function — and fall into an infinite loop when `main()` returns; see **Listing 2**.

The diagram in **Figure 4** illustrates how `_reset()` initializes `.data` and `.bss`.

The `firmware.bin` file is just a concatenation of the three sections: `.vectors` (IRQ vector table), `.text` (code) and `.data` (data). Those sections were built according to the linker script: `.vectors` lies at the very beginning of flash, `.text` follows immediately after, and `.data` lies far above. Addresses in `.text` are in the flash memory region,

and addresses in `.data` are in the RAM region. If some function has an address, e.g. `0x8000100`, then it is located exactly at that address in flash. But, if the code accesses some variable in the `.data` section by its address, e.g. `0x20000200`, then there is nothing at that address, because, at boot, the `.data` section in the `firmware.bin` file resides in flash! That's why the startup code must relocate the `.data` section from the flash memory region to the RAM region.

Now we are ready to produce the full firmware file, `firmware.elf`:

```
$ arm-none-eabi-gcc -T link.ld -nostdlib main.o -o
firmware.elf
```

Let's examine sections in `firmware.elf` — see **Listing 3**.

Now we can see that the `.vectors` section will reside at the very beginning of flash memory at address `0x8000000`, then the `.text` section right after it, at `0x80001ac`. Our code does not create any variables, so there is no data section.

Flash Firmware

We're ready to flash this firmware! First, extract the sections from `firmware.elf` into a single, contiguous binary blob:

```
$ arm-none-eabi-objcopy -O binary firmware.elf
firmware.bin
```

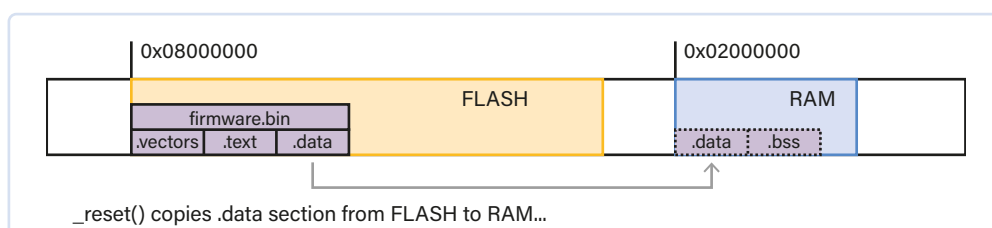


Figure 4: The diagram visualizes how `_reset()` initializes `.data` and `.bss`.



Listing 3: Sections in firmware.elf

000011000000

```
$ arm-none-eabi-objdump -h firmware.elf
...
Idx Name              Size      VMA       LMA       File off  Algn
  0 .vectors            000001ac  08000000  08000000  00010000  2**2
CONTENTS, ALLOC, LOAD, DATA
  1 .text                00000058  080001ac  080001ac  000101ac  2**2
CONTENTS, ALLOC, LOAD, READONLY, CODE
...
```

Then use *st-link* utility to flash *firmware.bin*. Plug your board into the USB, and execute:

```
$ st-flash --reset write firmware.bin 0x8000000
```

Done! We've flashed firmware that does nothing.

Makefile: Build Automation

Instead of typing those compilation, linking, and flashing commands, we can use the *make* command line tool to automate the whole process. The *make* utility uses a configuration file called *Makefile*, where it reads instructions on how to execute actions. This automation is great because it also documents the process of building firmware, compilation flags used, etc.

There is a great *Makefile* tutorial [5]. For those new to *make*, I suggest taking a look. Below, I list the most essential concepts required to understand our simple bare-metal *Makefile*. Those who already familiar with *make*, can skip this section.

The *Makefile* format is simple:

```
action1:
    command ... # Comments can go after hash symbol
    command .... # IMPORTANT: command must be
preceded with the TAB character
action2:
    command ... # Don't forget about TAB. Spaces
won't work!
```

Now, we can invoke *make* with the action name (also called *target*) to execute a corresponding action:

```
$ make action1
```

It is possible to define variables and use them in commands. Also, actions can be the names of files that need to be created:

```
firmware.elf:
    COMPILATION COMMAND .....
```

Also, any action can have a list of dependencies. For example, *firmware.elf* depends on our source file, *main.c*. Whenever the *main.c* file changes, the *make build* command rebuilds *firmware.elf*:

```
build: firmware.elf
```

```
firmware.elf: main.c
    COMPILATION COMMAND
```

Now we are ready to write a *Makefile* for our firmware. We define a *build* action /target:

```
CFLAGS ?= -W -Wall -Wextra -Werror -Wundef -Wshadow
-Wdouble-promotion \ -Wformat-truncation -fno-common
-Wconversion \ -g3 -Os -ffunction-sections -fdata-
sections -I. \ -mcpu=cortex-m4 -mthumb -mfloat-
abi=hard -mfpv4-sp-d16 $(EXTRA_CFLAGS)
LDFLAGS ?= -Tlink.ld -nostartfiles -nostdlib --specs
nano.specs -lc -lgcc -Wl,--gc-sections -Wl,-Map=$@.map
SOURCES = main.c
build: firmware.elf
firmware.elf: $(SOURCES)
    arm-none-eabi-gcc $(SOURCES) $(CFLAGS) $(LDFLAGS)
    -o $@
```

There, we define compilation flags. The *?* represents a default value; we could override them from the command line like this:

```
$ make build CFLAGS="-O2 ...."
```

We specify *CFLAGS*, *LDFLAGS*, and *SOURCES* variables. Then we tell *make*: if you're told to build, then create a *firmware.elf* file. It depends on the *main.c* file, and to create it, start *arm-none-eabi-gcc* compiler with the given flags. Special variable *\$@* expands to a target name — in our case, *firmware.elf*.

Let's call *make*:

```
$ make build arm-none-eabi-gcc main.c -W -Wall
-Wextra -Werror -Wundef -Wshadow -Wdouble-promotion
-Wformat-truncation -fno-common -Wconversion
-g3 -Os -ffunction-sections -fdata-sections -I.
-mcpu=cortex-m4 -mthumb -mfloat-abi=hard -mfpv4-
sp-d16 -Tlink.ld -nostartfiles -nostdlib --specs
nano.specs -lc -lgcc -Wl,--gc-sections -Wl,-
Map=firmware.elf.map -o firmware.elf
```

If we run it again:

```
$ make build
make: Nothing to be done for 'build'.
```


Listing 4: Section of Blinky LED main.c file

```
#include <inttypes.h>
#include <stdbool.h>
#define BIT(x) (1UL << (x))
#define PIN(bank, num) (((bank) - 'A') << 8) | (num))
#define PINNO(pin) (pin & 255)
#define PINBANK(pin) (pin >> 8)

struct gpio {
    volatile uint32_t MODER, OTYPER, OSPEEDR, PUPDR, IDR, ODR, BSRR, LCKR, AFR[2];
};
#define GPIO(bank) ((struct gpio *) (0x40020000 + 0x400 * (bank)))

// Enum values are per datasheet: 0, 1, 2, 3
enum { GPIO_MODE_INPUT, GPIO_MODE_OUTPUT, GPIO_MODE_AF, GPIO_MODE_ANALOG };

static inline void gpio_set_mode(uint16_t pin, uint8_t mode) {
    struct gpio *gpio = GPIO(PINBANK(pin)); // GPIO bank
    int n = PINNO(pin); // Pin number
    gpio->MODER &= ~(3U << (n * 2)); // Clear existing setting
    gpio->MODER |= (mode & 3) << (n * 2); // Set new mode
}
```

The `make` utility examines modification times for the `main.c` dependency and `firmware.elf` — and does not do anything if `firmware.elf` is up to date. But, if we change `main.c`, then the next `make build` will recompile:

```
$ touch main.c # Simulate changes in main.c
$ make build
```

Now, what's left is the `flash` target:

```
firmware.bin: firmware.elf
$(DOCKER) $(CROSS)-objcopy -O binary $< $@ flash:
firmware.bin
st-flash --reset write $(TARGET).bin 0x8000000
```

That's it! Now, the `make flash` terminal command creates a `firmware.bin` file, and flashes it to the board. It will recompile the firmware if `main.c` changes, because `firmware.bin` depends on `firmware.elf`, and it, in turn, depends on `main.c`. So, now the development cycle would be these two actions in a loop:

```
# Develop code in main.c
$ make flash
```

It is a good idea to add a clean target to remove build artifacts:

```
clean:
rm -rf firmware.*
```

Complete project source code can be found in Step 0 minimal folder [6].

Blinky LED

Now, as we have the whole build / flash infrastructure set up, it is time to teach our firmware to do something useful. Something useful is, of course, blinking an LED. A Nucleo-F429ZI board has three built-in LEDs. In the Nucleo board user manual [2], section 6.5 shows that we can see to which pins the built-in LEDs are connected:

- PB0: green LED
- PB7: blue LED
- PB14: red LED

Let's modify the `main.c` file and add our definitions for PIN, `gpio_set_mode()`. In the `main()` function, we set the blue LED to output mode, and start an infinite loop. First, let's copy the definitions for the pins and GPIO we discussed earlier. Note that we also add a convenience macro, `BIT(x)` — see **Listing 4**.

Some microcontrollers, when they are powered, have all their peripherals powered and enabled automatically. STM32 MCUs, however, have their peripherals disabled by default in order to save power. To enable a GPIO peripheral, it should be enabled (clocked) via the RCC (Reset and Clock Control) unit. In Reference Manual section 7.3.10 we find that the `AHB1ENR` (AHB1 peripheral clock enable register) is responsible for turning GPIO banks on or off. First, we add a definition for the entire RCC unit:

```
struct rcc {
    volatile uint32_t CR, PLLCFGR, CFGR, CIR,
    AHB1RSTR, AHB2RSTR, AHB3RSTR, RESERVED0, APB1RSTR,
    APB2RSTR, RESERVED1[2], AHB1ENR, AHB2ENR, AHB3ENR,
    RESERVED2, APB1ENR, APB2ENR, RESERVED3[2], AHB1LPENR,
```

```
AHB2LPENR, AHB3LPENR, RESERVED4, APB1LPENR,
APB2LPENR, RESERVED5[2], BDCR, CSR, RESERVED6[2],
SSCGR, PLLI2SCFGR;
};
#define RCC ((struct rcc *) 0x40023800)
```

In the `AHB1ENR` register's documentation, we see that bits from 0 to 8, inclusive, set the clock for GPIO banks GPIOA–GPIOE:

```
int main(void) {
    uint16_t led = PIN('B', 7); // Blue LED
    RCC->AHB1ENR |= BIT(PINBANK(led));
    // Enable GPIO clock for LED
    gpio_set_mode(led, GPIO_MODE_OUTPUT);
    // Set blue LED to output mode
    for (;;) asm volatile("nop"); // Infinite loop
    return 0;
}
```

Now, what is left to do is to find out how to set a GPIO pin to on or off, and then modify the main loop to set an LED pin on, delay, off, delay. Looking at reference manual section 8.4.7, we see that register `BSRR` is responsible for setting the voltage high or low. The low 16 bits are used to set the `ODR` register (i.e. set pin to high), and the high 16 bits are used to reset the `ODR` register (i.e. set pin to low). Let's define an API function for that:

```
static inline void gpio_write(uint16_t pin, bool val) {
    struct gpio *gpio = GPIO(PINBANK(pin));
    gpio->BSRR |= (1U << PINNO(pin)) << (val ? 0 : 16);
}
```


Next, we need to implement a delay function. We do not require an accurate delay right now, so let's define a function, `spin()`, that just executes a NOP instruction a given number of times:

```
static inline void spin(volatile uint32_t count) {
    while (count--) asm("nop");
}
```

Finally, we're ready to modify our main loop to implement LED blinking:

```
for (;;) {
    gpio_write(pin, true);
    spin(999999);
    gpio_write(pin, false);
    spin(999999);
}
```

The complete project source code can be found in Step 1 blinky folder [7]. Run `make flash` and enjoy the flashing blue LED!

In the second part of this article, we will have a look at UART output, debugging, a web server implementation, automatic tests, and more. Stay tuned! 

220665-01

About the Author

Sergey Lyubka is an engineer and entrepreneur. He holds an MSc in Physics from Kyiv State University, Ukraine. Sergey is director and co-founder of Cesanta, a technology company based in Dublin, Ireland (Embedded Web Server for electronic devices: <https://mongoose.ws>). His passion is bare-metal embedded network programming.

Questions or Comments?

Do you have technical questions or comments about this article? Email the author at sergey.lyubka@cesanta.com or contact Elektor at editor@elektor.com.



Related Products

- > **Dogan Ibrahim, Nucleo Boards Programming with the STM32CubeIDE, Elektor**
<https://elektor.com/19530>
- > **Dogan Ibrahim, Programming with STM32 Nucleo Boards, Elektor**
<https://elektor.com/18585>

WEB LINKS

- [1] Reference manual RM0090 for STM32F429: <https://bit.ly/3neE7S7>
- [2] Nucleo-144 Board User manual (UM1974): <https://bit.ly/3oIBXKZ>
- [3] Article on GitHub: <https://github.com/cpq/bare-metal-programming-guide>
- [4] Contents from link.ld file: <https://github.com/cpq/bare-metal-programming-guide/blob/main/step-0-minimal/link.ld>
- [5] Makefile tutorial: <https://makefiletutorial.com/>
- [6] Step 0 minimal demo program: <https://github.com/cpq/bare-metal-programming-guide/blob/main/step-0-minimal>
- [7] Step 1 blinky demo program: <https://github.com/cpq/bare-metal-programming-guide/blob/main/step-1-blinky>
- [8] .bss [Wikipedia]: <https://en.wikipedia.org/wiki/.bss>



Siglent SDM3045X Multimeter

By Philippe Demerliac (France)

A typical handheld multimeter serves its niche well, and is always convenient to have on hand. A bench multimeter, however, offers many more capabilities that make life in the lab much easier. Siglent's SDM3045X bench multimeter is one such device, and I review it here.

I already own several Siglent devices, which I enjoy for their excellence. However, I was missing a bench multimeter from their vast range of instruments. Now that I've filled this gap with the SDM3045X, I can share my impressions.

Siglent offers three multimeter families: SDM3045X, SDM3055, and SDM3065X. These models have similar functionalities but differ in resolution. The 3045 has 4½ digits (60,000 counts), the 3055 has 5½ digits (240,000 counts), and the 3065 has 6½ digits (2,200,000 counts). The 3055 and 3065 also offer better low-sensitivity performance: 200 mV / 200 µA, versus 600 mV / 600 µA for the 3045. In addition, the 3055 and 3065 offer the SC option, which allows multiple programmable measurement inputs. These models share the same enclosure and ergonomics. The prices, of course, vary with resolution.

Resolution and Accuracy

Let's talk a little about resolution. It is expressed as a number of significant digits, or counts. The number of counts indicates the number of distinct values that can

be displayed for a given range. For example, at 600 mV, its highest-resolution range, the 3045 can display increments of 10 µV/count (0.6 V / 60,000 count). Each additional range up — 6 V, 60 V, 600 V, 1000 V — brings with it a reduction in resolution, with the 1000 V range having 100 mV/count. (Note that the highest range is 1000 V rather than 6000 V, for safety reasons.)

On higher-resolution models, there are more significant digits. Be careful not to confuse resolution with accuracy. If we read a measurement of 1.0000 V on the screen, are we certain that the voltage is 1 V to an accuracy of 100 µV? No, because even properly calibrated specifications indicate a maximum error of 0.06% ±8 digits. So, the actual voltage could be anything between 0.9986 V and 1.0014 V, these limits being maxima.

Does this mean that the lowest figures are useless? No, because they make it possible to compare measurements and to observe their progress over time. Going back to our SDM3045X, is 60,000 counts enough, or do we need to select a different model? The answer to this question obviously depends on usage. What I can say is that, for an amateur or even a pro user, this resolution is more than enough, even often superfluous. I have used and still use multimeters with lower resolutions without this being a problem in 90% of cases.

You can easily download the model datasheet from Siglent [1] and see the maximum resolution and error for each type of measurement and, for each range, the maximum resolution and error guaranteed by the manufacturer. Speaking of accuracy, the multimeters come with a calibration certificate proving that the device was within the advertised limits, often better.



Siglent says in this regard:

"SIGLENT has determined that the factory calibration of this instrument is not significantly affected by storage of up to 180 days prior to first use. The calibration interval must start when the device is put into service or 180 days after the 'calibration date' on the certificate received with the device."

Should these multimeters be recalibrated periodically? In a professional setting, it is advisable to check if you want to meet certain standards, but for an amateur, this is optional. From experience, modern materials do not drift much with time. If you're able to do so, it is advisable to check the accuracy of the device only periodically.

There are approved metrology centers that can perform these calibrations, but it is a relatively expensive operation compared to the price of the device. Even if Siglent is not very verbose regarding the calibration, there are hacks on the internet to do it, but this still presupposes having the appropriate standard sources. In summary, this is not usually a concern, and your multimeter will serve you faithfully for many years.

First Look at the SDM3045X

The SDM3045X is a bench multimeter, and so, although it is transportable, is not really intended for mobile use in the field. It is already not autonomous and requires a mains power supply, then it must be placed on a table or at least a stable support surface to use it. Over stand-alone models, bench multimeters generally offer better and additional features, such as connectivity, 4-wire measurement, richer display, etc... You can also place them securely right in front of you, making their use obviously more comfortable, and mains supply relieves the hassle of changing the batteries or of periodic recharging.

The SDM3045X can measure voltages and currents in DC and AC (RMS) up to 100 kHz (perfect for LF!), resistances (2 wires or 4 wires for very low resistances), continuity, diode only, capacitors, and temperature with different sensors (it incorporates cold junction compensation for thermocouples). This covers most current needs. It also makes it possible to memorize the values read, create statistics, and see alarms in the case of over-limit measurements.

As with other Siglent devices, it comes in a well-designed package that protects it perfectly against shocks. I advise you to keep it treasured. The device comes with a power cord, two flexible test leads of excellent quality,

a USB A/B cable for connection to a PC, a calibration certificate, and a basic user manual in English. Detailed manuals, including in other languages, can be downloaded from the Siglent website.

The device gives off an air of quality, and the finish is impeccable. Soft keys offer a pleasant touch. The power button is not an "actual" switch, so the device is always on standby. Personally, I prefer a real disconnection from the mains, and disconnect everything when I do not use these devices, via a socket with a master switch.

The screen is very readable. When starting up, it takes a few tens of seconds to initialize. In any case, it is advisable not to turn these devices on/off for short periods, the maximum accuracy only being guaranteed after a warm-up period (10 to 30 min). But we can, of course, use it before that. In contrast to the superior models with their somewhat noisy fans, the SDM3045X has no fans and is therefore silent, apart from a few discrete relay clicks.

The rear panel (**Figure 1**) offers two BNC sockets, an input to trigger the measurements via an external signal, and an output that indicates that the measurement is complete. These plugs are especially useful for automatic benches, but are rarely utilized in everyday use. There is also an RJ45 LAN connector and a USB-B plug to connect the device to a PC. A Kensington-type security slot is also provided on the SDM3045X for securing an anti-theft cable. The ammeter's 10 A fuse is also accessible for easy changing in case of an overcurrent event.

Figure 1: Rear panel.



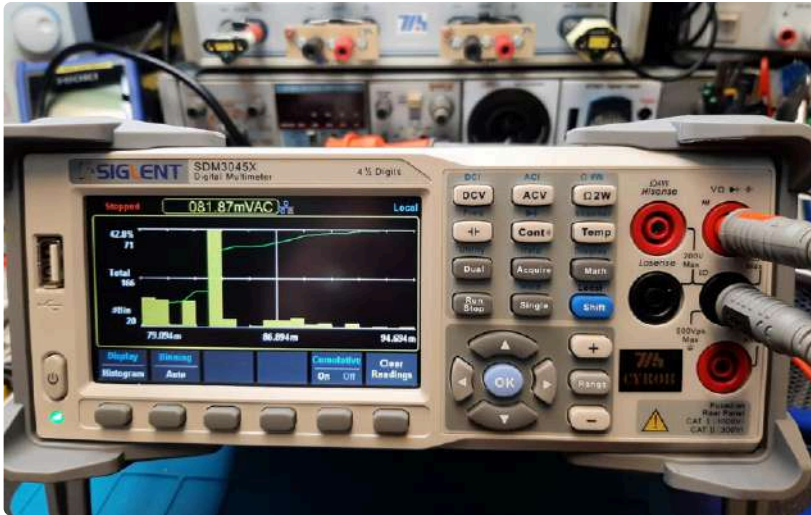


Figure 2: Histogram mode.

Initially, I advise you to check that the firmware is up to date. Siglent often updates its devices, the updates being easily downloadable on their site. Updates are done via a USB stick plugged into the front panel.

Basic Use

Basic use is simple and intuitive. You can easily switch from one measurement mode to another via the buttons on the front. It is possible to change the measurement speed, which affects the temporal resolution. In many cases, the fast mode will be sufficient, with an almost instantaneous response time. The automatic range change also greatly reduces the necessary handling. However, the range can easily be selected manually at any time.

For AC voltage and current measurement, it offers 100 kHz bandwidth and true RMS measurement. Like most multimeters, however, a correction of the peak factor must be made manually for asymmetric signals; the manual gives all the details on this point.

In continuity or diode test mode, we can adjust the threshold of the beep, as well as its volume, which is a plus. In all measurements, we can set a “relative” mode to see the measurements with reference to a predefined value. You can also halt acquisition manually and memorize the last measurement at any time.

Interesting Features

The SDM3045X behaves like a basic multimeter, but it offers additional features that can make life easier. Here are the main ones:

- Different display modes: A bar graph lets you visualize the order of magnitude of the measurement and its variation, Histogram mode (Figure 2) shows the statistical distribution of the measurements graphically, allowing you to immediately see the most frequent values, and a curve mode displays the evolution of the measurement over time, even for periods as short as a second. This is very convenient for

observing the change in values over time. In all these modes, scales and limits can be calculated automatically or set manually.

- A statistical mode displays the mean, min, max, standard deviation, etc., of the measurements.
- The Dual display (Figure 3) allows you to monitor two parameters simultaneously, such as voltage and frequency.
- A dB/dBm mode enables relative measurements in dB or power measurements in dBm for a load impedance that needs to be set. (CAUTION: this setting does not change the input impedance of the multimeter; you must connect the desired load externally.)
- The Limit mode (Figure 4) visually indicates whether the measured value is between two adjustable limits, which is very convenient for repetitive circuit settings without having to interpret the values.
- The Probe Hold mode (Figure 5), one of my favorites, automatically memorizes consecutive stable measurements, making it convenient for sorting components.
- You can fine-tune the acquisition mode, sampling speed, and whether it is auto, manual, or triggered by an external trigger signal with a definable value and polarity.



Figure 3: Dual display.



Figure 4: Limit mode.



Figure 5: Probe Hold mode.

- Measured values and/or current settings can be stored on a USB drive for easy recall.
- The thermometer mode accepts different types of common thermocouples (with cold junction compensation) and resistive sensors. (Note: there is no temperature probe included with the device.)
- Built-in help reminds you of the proper usage and connections for measurements (in English).

Use with a PC

The device can be controlled by SPI, via USB or LAN Ethernet. LabVIEW drivers are available on the Siglent website. Protocol documentation is provided for operating the device. Additionally, Siglent offers the free Windows EasyDMM application (**Figure 6**), which makes it easy to control all Siglent multimeters, view measurements, and export them to CSV. This application, despite its somewhat outdated appearance, works very well.

Conclusion

I have never been disappointed by Siglent measuring devices, and this multimeter also made a great impression on me overall. It is practical and likely covers most of the needs of a troubleshooting or R&D lab. The price is somewhat higher than entry-level Asian models, but the quality of the product justifies it. The price-quality ratio is excellent, and, if you want to invest in a practical and efficient bench multimeter, I highly recommend it.

I particularly enjoyed:

- Ease of use.
- Overall quality and highly readable screen.
- Wealth of features offered.
- Silence (no fan).
- Value for money.
- Detailed documentation in French.
- Easily available updates.

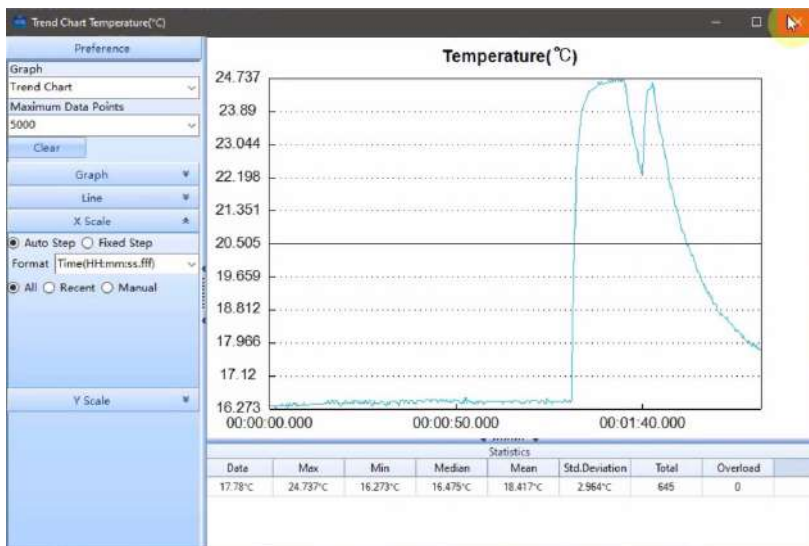


Figure 6: EasyDMM Windows Application

I did not enjoy:

- Soft power switch that only goes into sleep mode.
- The service manual: There's lack of clear information or useful instructions for calibration — there's a procedure using PHP scripts, but it is very complex. There are instructions to open the unit and check voltages, but no circuit diagram. In case of problems, you have to contact Siglent. ◀

230126-01

Questions or Comments?

Do you have any technical questions or comments about this article? Contact the author at info@cyrob.org or the Elektor team at editor@elektor.com.



About the Author

Philippe Demerliac, born in 1962, is a designer of electronic circuits for scientific purposes, with a lifelong passion for precision mechanics, measurement devices, metalworking, and science in general. Although his professional life took a turn toward software, he remains dedicated to the field of electronics. In an effort to give back to the community, Philippe created the cyrob.org website and, later, a French YouTube channel: [youtube.com/@Cyrob-org](https://www.youtube.com/@Cyrob-org)

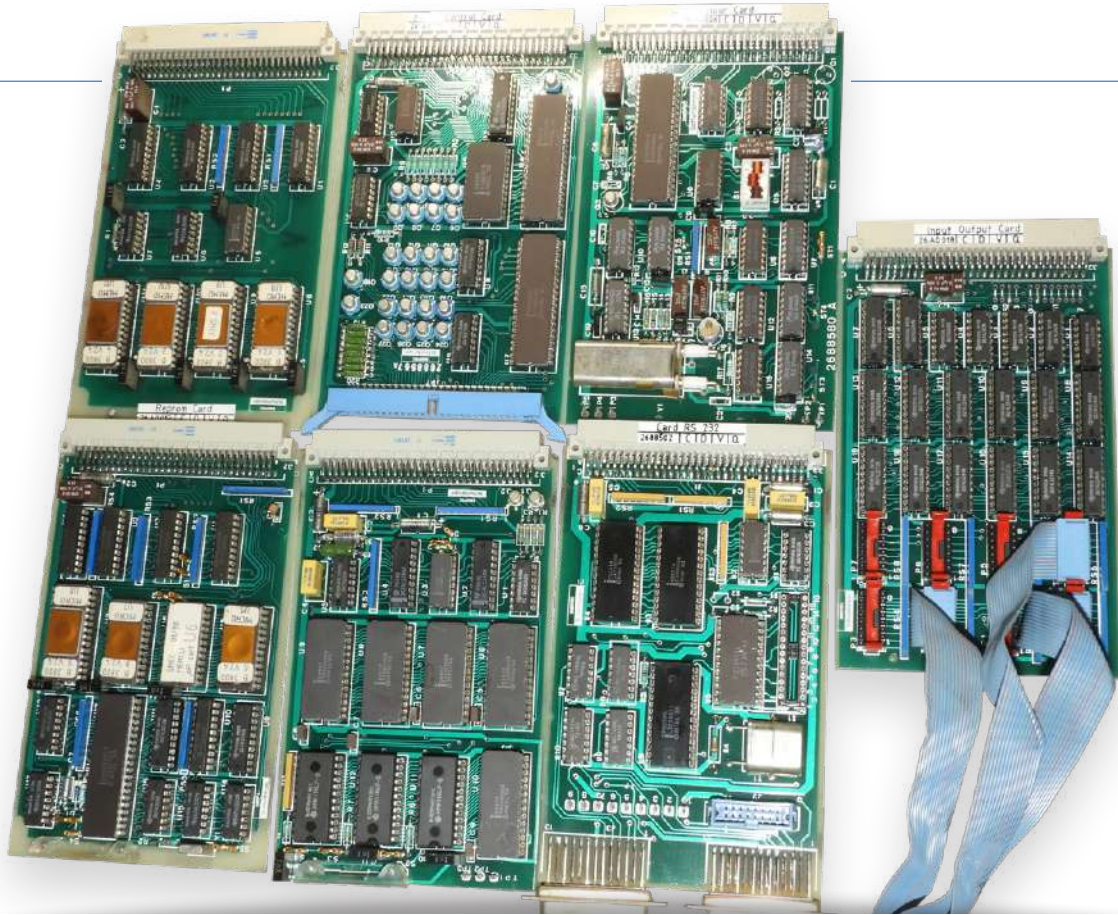


Related Products

- **Siglent SDM3045X Digital Multimeter**
<https://elektor.com/17892>

WEB LINKS

[1] Siglent SDM3045X datasheet: <https://bit.ly/40iCh00>



Microprocessors for Embedded Systems

Peculiar Parts, the Series

By David Ashton (Australia)

Embedded engineers and hobbyists are spoiled for choice these days regarding computing power, thanks to the vast array of microcontrollers on offer. But, it wasn't always like this, so let's delve into the embedded systems of the past...

Today, even low-end microcontrollers such as the 8-pin AVR ATtiny have up to 8 KB of flash memory and up to 512 bytes of EEPROM and SRAM. Then there's the peripheral offering: two timers/counters, serial interface, four-channel ADC, watchdog timer, and an analog comparator. It will execute your code at 10 MHz and support a wide input voltage range.

By comparison, life for early embedded systems developers was hard. They relied upon microprocessors without any on-chip memory (apart from the registers) or peripherals. Instead, all that good stuff had to be added to form a complete system.

This meant attaching other devices to the microprocessor's bus — very peculiar.

Typically, you'd have 16 bits for addresses, providing space for 64 KB of memory, and 8 bits for data. Then there would be various control signals, such as read/write and timing signals. Any peripheral chips you connected would mostly have a single function. For memory, you'd include a ROM (pre-programmed or EPROM) and RAM, which, if you wanted any decent amount, had to be dynamic. This required fiendishly complicated refresh timings, often needing a separate chip. Peripherals were also added separately. Timers, counters, input/output ports, UARTs for serial communication, ADCs, CRT controllers (which had outputs to drive displays), and so on. Each chip was



Figure 1: Dating back to the mid-1980s, this board features an 8085 microprocessor. Combined with six other boards (see article header), it became as fully featured as today's microcontrollers!

a 24- to 40-pin dual inline monster, usually matched to the microprocessor from the same manufacturer, running at a maximum of 4 MHz. But, the fun didn't end there. Next came the basic logic ICs for decoding and buffering, known as glue logic. Programmable logic chips, such as GALs, PALs, and ULAs, were popular for this task, but sometimes manufacturers poured such functionality into dedicated custom chips.

Intel started the ball rolling in 1974 with the venerable 8080, which could be said to herald the true start of the microprocessor age. It needed two support chips, the 8224 clock generator, and the 8228 bus controller. Unusual for today's engineers, it also needed both ± 5 V and ± 12 V supplies. Later iterations, such as the 8085 (Figure 1), only needed a $+5$ V power supply. However, it was more complex due to its strange bus-multiplexing scheme. This required extra glue logic, although a dedicated line of 8085 peripheral chips was available.

Programs were stored in UV-EPROMs during development — ultraviolet-erasable programmable read-only memories. Thanks to a little window above the die, the content could be cleared using an EPROM eraser — a box with a UV bulb, timer, and a slide-out tray for the EPROM. The UV light erased the contents in about 20–30 minutes. Afterward, it could be reprogrammed using an EPROM programmer plugged into your PC's serial or parallel port. How this was done before the advent of PCs, I don't know! However, I do recall seeing programmers with hex keypads typing data into each memory location...

Programming was done in machine code, entering hexadecimal values into each memory location. The other option was assembly language that used mnemonics

for the operations and labels for variables and code sections. Instructions such as "ADC B" meant "add the contents of Register B to Register A, with carry." Languages such as C were but a gleam in their creator's eye, though if you were lucky, you might get a BASIC interpreter. If your program didn't work or needed rewriting, you erased the EPROM and reprogrammed it. Once you had your final program, you could have it burned onto ROMs. While these were cheaper per device, you couldn't erase them!

Other manufacturers also got on the microprocessor bandwagon. Motorola had their 6800, MOS Technology the 6502, and National Semiconductor the SC/MP. Some ex-Intel designers formed Zilog, which offered the Z80 (Figure 2), a kind of deluxe 8080. Most of these chips were used in the first home computers, such as the Sinclair ZX80/81 and Spectrum (Z80), the Commodore 64 (6502 variant), and many others. Elektor's SC/MP project from 1978 is another good example [1].

In 1980, Intel introduced the 8051. This had a small amount of ROM (sometimes EPROM) and RAM on board, but also four 8-bit ports, a UART, and two counter/timers. With everything in a single device, it could be said that this is where the microcontroller era started. The rest, as they say, is history. The 8051 architecture is still in use today but with many more peripheral functions than its ancestors, and doing away with the buses and interconnect logic of the past.

The 8051 and Motorola's 68xx devices spawned the rich and versatile range of microcontrollers that we know and love today: AVR, PICs, and a host of Arm-based others, giving rise to boards like Arduino and Raspberry Pi. Intel parlayed the 8085 into the 16-bit 8086, used in the first IBM

PC XT, and then the 80286, 80386, 80486, and the Pentiums that are still used in PCs today. It should be noted that microprocessor-based SBCs, rather than microcontrollers, were still common until the 1990s.

When the 8080 came out, I was just 18 and remember reading about it in the hobby electronics magazines of the day. I was lucky enough to work on microprocessor equipment early in my career, which was exciting. My roles were focused on a bit of programming and lots of fault-finding. I've been fortunate to follow this technology throughout my career, and it's always fascinated me. Compared to today's microcontrollers, the old microprocessors were definitely peculiar, but they were still lots of fun to work on. ◀

230047-01



Figure 2: Another workhorse of early embedded systems was Zilog's Z80. Here it is accompanied by a serial I/O controller (SIO) and counter/timer circuit (CTC).

WEB LINKS

[1] J. Buiting, "The Elektor SC/MP Computer," Elektor Circuit Special 2022: <https://elektormagazine.com/220195-01>

Microcontroller Documentation Explained

(Part 3)

Block Diagrams and More

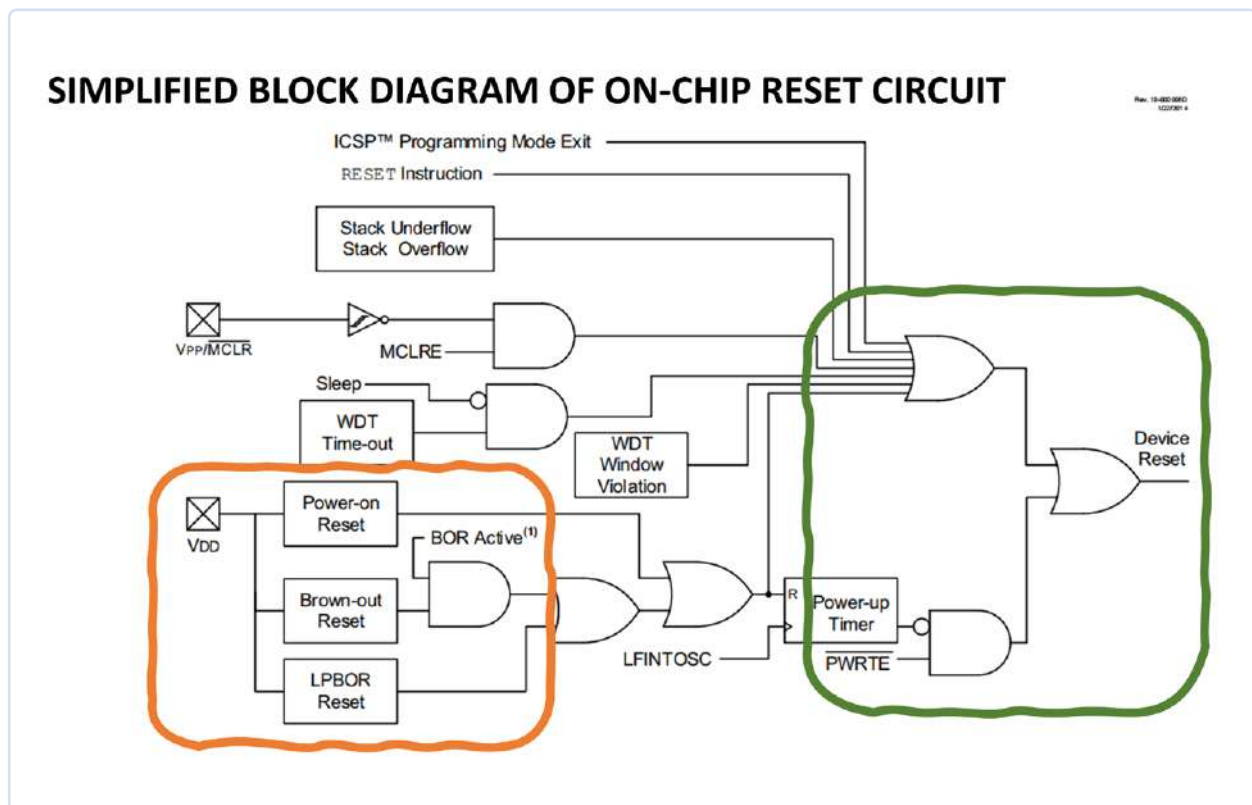


Figure 1: The reset circuit provides very few configuration options, with some reset sources being linked to the VCC power supply pin.
(Source: Microchip Technology)

By Stuart Cording (Elektor)

In the previous parts of this series [1], we covered register functionality and took a look at the clock block diagram. Here we provide more block diagrams and investigate where you can find the rest of the required documentation.

Rev. 10-000017E
6/15/2018

little more than squares. Some, such as the prescaler and postscaler, are self-explanatory. The **SYNC** block seems self-explanatory, but will probably require the developer to read the accompanying text in detail to fully appreciate its function. Thus, diagrams and text often go hand-in-hand in datasheets. Sometimes it is necessary to write some test code to really understand how a peripheral works.

Building a Board

At some point, the design needs to move to a PCB if you're planning on mass-producing your design. Most computer-aided design (CAD) software will have the schematic blocks for your processor, along with the 'land pattern' for various packages for the PCB design. However, if they are missing, the datasheet will probably include technical drawings for each package and the associated land pattern. In this case, they can be found on *page 639*.

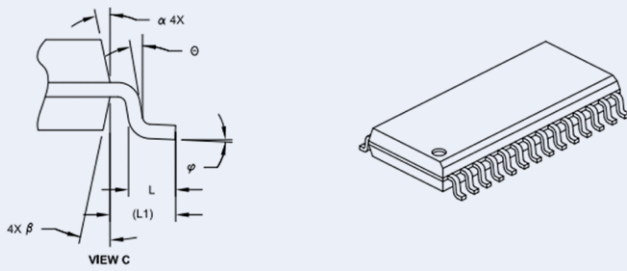
The SOIC package shown here can be found on *page 645* and *646* (**Figure 3**). For PCB designers, perhaps the most useful data today is the height of the package if the design is limited in volume (they will already know the width and length). Many packages today also feature an integrated “metal slug” that needs to be soldered to the PCB to aid heat dissipation. This is not the case here but, if it were, guidance on how to connect it would be given. You may also need to check whether it is connected to the ground of the device or perhaps another pin.

What Is Only Sometimes in a Microcontroller Datasheet?

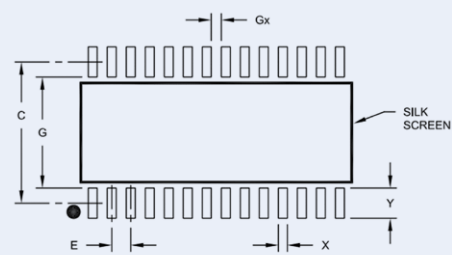
As you will now be aware, there is a lot of information in a datasheet. However, there are a few things that may be left out because it would make the datasheet too long, or the information is common to many devices and it deserves its own documentation. These can include:

- Processor instruction set: If the number of instructions is low (fewer than 60–70), they may all be included with an explanation in the datasheet. If not, there is probably a

elektor July & August 2023 113



Units		MILLIMETERS		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N	28		
Pitch	e	1.27 BSC		
Overall Height	A	-	-	2.65
Molded Package Thickness	A2	2.05	-	-
Standoff	§	A1 0.10	-	0.30
Overall Width	E	10.30 BSC		
Molded Package Width	E1	7.50 BSC		
Overall Length	D	17.90 BSC		
Chamfer (Optional)	h	0.25	-	0.75
Foot Length	L	0.40	-	1.27
Footprint	L1	1.40 REF		
Lead Angle	θ	0°	-	-
Foot Angle	φ	0°	-	8°
Lead Thickness	c	0.18	-	0.33
Lead Width	b	0.31	-	0.51
Mold Draft Angle Top	α	5°	-	15°
Mold Draft Angle Bottom	β	5°	-	15°



Units		MILLIMETERS		
Dimension Limits		MIN	NOM	MAX
Contact Pitch	E	1.27 BSC		
Contact Pad Spacing	C	9.40		
Contact Pad Width (X28)	X	0.60		
Contact Pad Length (X28)	Y	2.00		
Distance Between Pads	Gx	0.67	-	-
Distance Between Pads	G	7.40	-	-

Figure 3: Package drawings provide all the dimensions of the package (left) along with the recommended 'land pattern' for the PCB (right). (Source: Microchip Technology)

separate document where each instruction is explained in detail.

- Code examples: These appear more often when the microcontroller is still commonly programmed in assembler. Examples in high-level languages such as C make little sense, since the compiler will define the precise instructions used.
- Circuit examples: These are most common in association with functionality unique to the microcontroller selected, such as the oscillator or the power supply, or slightly more complex interfaces that have specific requirements on signal loading, such as USB. However, they are more likely to be covered in more detail in an application note on the subject.

What Is Not Included in a Microcontroller Datasheet?

The following items are normally excluded from the datasheet, being found instead in other documentation. This is usually because it covers a topic that is common to a wide range of microcontroller devices.

- Flashing of microcontroller memory: This is normally covered separately as a topic for those focused on programming microcontrollers in mass production.
- Use of development tools: The compiler, use of an Integrated Development Environment (IDE), and debug tools have their own documentation.
- Detailed explanation of processor core: This will often be covered in a separate document, especially for 16-bit and 32-bit cores.
- Detailed explanation of complex peripherals: USB, graphics interfaces, and Ethernet peripherals are typically covered in separate documents as the inclusion of each could double the size of the microcontroller's datasheet.
- Errata: Any errors in the documentation, or workarounds to resolve silicon bugs that won't be fixed, are covered here.

What Other Microcontroller Documentation Is Offered?

In addition to the datasheet, the following documents are commonly made available:

- "Family Reference Manual:" While the datasheet tells you precisely what one microcontroller can do, such documents provide a higher-level overview of what all microcontrollers in a family of devices can do.
- Application notes: These go into a lot more detail on how to use specific peripherals, or a group of peripherals, to implement an application or interface. The datasheet may explain how to use the CAN (controller area network) peripheral, but the application note will explain how to use it as part of a CAN network, providing guidance on higher-level software protocols, and the selection of suitable transceivers.
- Programming specification: For programming in mass-production environments, this explains in detail the voltages and timings required, and any protocol used for the programming interface.

How to Consume Lots of Written Material

Unfortunately, there is no way to quickly consume all the data associated with a microcontroller and all of its tools. If you're a beginner, it is probably best to read the datasheet alongside a book or article on the microcontroller you're trying to use. Elektor not only has books for those interested in PIC microcontrollers; there are also starter books for the STM32 [4] and ARM-based microcontrollers [5] in general. Or, why not try a simple MSP430 project [6]?

The combination of practical examples coupled with two types of written explanation (a book/article and the datasheet itself) will help. If you're more advanced, the best approach is to focus on the sections that cover the processor core, the clock tree, and the reset block, followed by the peripherals you intend to use.

Then, you'll need to acquaint yourself with the documentation for the compiler tool chain. Also make liberal use of software libraries and examples to build up your understanding, and post questions on forums if you need further help.

Datasheets can be challenging to read and understand, and aren't the most exciting form of literature. However, they are (mostly) accurate and a technical description of functionality (the errata can be used to determine the confidence you afford the datasheet). Stick with them and, over time, you'll develop a good understanding of how they are constructed and worded. ◀

230286-01

Questions or Comments?

Do you have questions or comments about this article? Email the author at stuart.cording@elektor.com.



Related Products

- > **T. Hanna, *Microcontroller Basics with PIC*, Elektor 2020**
<https://elektor.com/19188>
- > **A. Pratt, *Programming the Finite State Machine*, Elektor 2020**
<https://elektor.com/19327>

WEB LINKS

- [1] Microcontroller Documentation series: <https://elektormagazine.com/tags/microcontroller-documentation>
- [2] Microchip Technologies PIC16F18877 Datasheet: <https://microchip.com/en-us/product/PIC16F18877>
- [3] The D flip-flop: https://en.wikipedia.org/wiki/Flip-flop_%28electronics%29#D_flip-flop
- [4] Starter books for the STM32: <https://elektor.com/programming-with-stm32-nucleo-boards-e-book>
- [5] ARM-based microcontrollers: <https://elektor.com/embedded-in-embedded>
- [6] Simple MSP430 project: <https://elektormagazine.com/labs/elektorpost-no-1-led-earring>



Farnell
AN AVNET COMPANY

TEST & MEASUREMENT SOFTWARE SOLUTIONS

We now deliver your digital software solution to go with your hardware.

Choose your T&M software and receive within 72 hrs of purchase by email.





Low-Power LoRa Weather Station

Build a Long-Range Weather Station by Yourself

By Edward Ringel (USA)

When my aged remote temperature and humidity station finally failed, I replaced it with a system of my own design. A battery-powered outdoor sensor and a battery-powered indoor display are connected via LoRa. Developing a reliable device able to withstand a winter in the northeast US was not easy! But, read for yourself.



Figure 1: Snow-covered sensor hanging near our driveway.

The minimum system functionality was to develop a remote, battery-powered outdoor sensor (**Figure 1**) and a battery-powered indoor display, also equipped with a sensor. As the project developed, I added a very simple web server.

To do this, I wanted to use the Arduino environment and its many high-quality libraries. To make the project simple, I planned to use off-the-shelf components that could be hand soldered. Finally, minimizing power consumption was a priority. Initially unanticipated, effective weatherization of the outdoor sensor was surprisingly challenging.

General considerations let me determine which hardware / which boards should be used — read more below about it and the difficulties I encountered. First of all, an outdoor station, an indoor station and a base station are needed. The latter collects the data from the two measuring stations and passes them on to a computer. There-

fore, all three units need at least one micro-controller board and a radio unit to communicate with each other. The two measuring stations for outside and inside also need sensors and a timer control, which will be discussed later. Below is a brief overview of the central “building blocks” of the three stations:

- Outdoor station: Artemis Nano, LoRa radio module, timer, sensors
- Indoor station: Artemis Thing Plus, LoRa radio module, timer, sensors, display
- Base or home station: Raspberry Pi Pico, Arduino Mini, LoRa radio module.

Power Considerations

Measurements of temperature, humidity, and barometric pressure are made periodically. Power consumption is minimized by having the remote measuring device in a low-power or sleep state, waking up, taking and transmitting readings, and returning to a low-power state. The duty cycle



Figure 2: My ePaper display shows the temperature (in °F), relative humidity, and barometric pressure uncorrected/not normalized. The “i” and “o” subscripts refer to inside and outside. Only outside pressure is displayed.

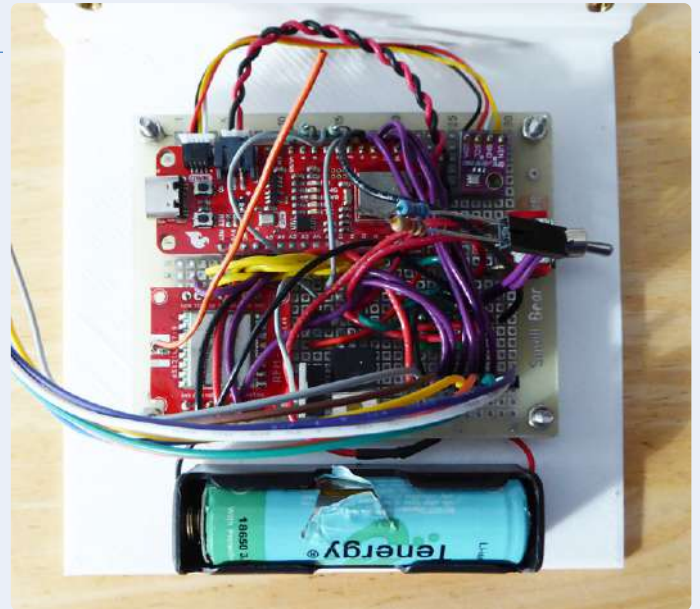


Figure 3: Early iteration of display board. Early version of display circuitry, prior to Artemis board modification and added TPL5111. Optocouplers, MOSFETs, TPL5110, switches for recharging, yikes!!

of power consumption is represented by T_{ON} / T_{TOT} . The effective power consumption is governed by:

- Power consumption of the device while on and processing instructions or data.
- Fraction of time the device is on to collect and send (or display) data — minimizing the duty cycle significantly affects overall power consumption.
- Power consumption of the device between measurements. Ideally, there would be no power drawn between sampling events, but that is not a realistic expectation.

I tested the baseline power consumption of multiple MCU boards by powering up the board without running any programs or attaching peripherals. See **Table 1** for the results (these may not represent power consumption after fine-tuning as shutting off peripheral interfaces, slowing the clock, etc.). In light of these findings, I decided to use the Artemis boards from Sparkfun.

Power utilization was also a factor in designing the indoor display station, but display choice was easy: ePaper has the lowest power consumption. This technology draws current only upon display refresh (that’s why the ePaper-equipped Kindle can display

a graphic while inactive and consuming no power). WaveShare has a nice 4.2” ePaper display. Instructions were hard to follow, and I needed to generate some fonts, but as **Figure 2** ultimately demonstrates, the display worked well.

I used Bosch BME280 sensors for measurements. These units measure temperature, humidity, and barometric pressure (T/H/P) and have robust I²C libraries. Their current draw is negligible.

A timing mechanism controlled the duty cycle. The MCU board, radio, and sensor had to be turned on and off — it would make no sense to utilize an efficient MCU sleeping at 20 µA while the radio drew 20 mA. Rather than coding MCU sleep and radio sleep, I switched power on and off to the entire apparatus.

The 3.3 V regulator on the Artemis board has a maximum power output of 600 mA, enough to power all components of the inside display as well as of the outside sensor / radio. Because I needed to turn off the power to the boards, the best solution was an ultra-low power timer chip. These devices have an internal timing circuit, with the interval set by a single external resistor, and are used with an MCU.

When the timer turns ON, current flow is enabled. When the MCU has completed its task, it sends a DONE message to the timer, and current flow is disabled. I worked with the TPL5110 chip on a homemade board and on the Adafruit breakout board, both in combination with an early version of the display board (**Figure 3**) extensively. My unsuccessful adventures with this little guy are documented in [1].

Table 1. Boards and Power.

Board	Power Consumption (mA)
Teensy 3.5	74.5
Raspberry Pi Pico	20.4
Teensy 4.1	92.6
ESP32 Adafruit Huzzah Feather	125.0
Artemis Thing Plus	8.5
Generic ESP32-WROOM	70.3

Power consumption of various MCUs powered up, supplied with 5 V to V_{in} , but not running any programs. Power consumption is governed not only by the MCU, but other board components (LEDs, voltage regulators, external memory, wireless configuration, etc.).



I ultimately used a TPL5111 connected to the enable pin of the 3.3 V regulator on the Artemis boards (**Figure 4**). This implementation required the removal of SMD pull-up resistor R1 on the Artemis Nano (**Figure 5**), or R3 on the Artemis Thing Plus (**Figure 6**) but allowed me to have control over all power on the board.

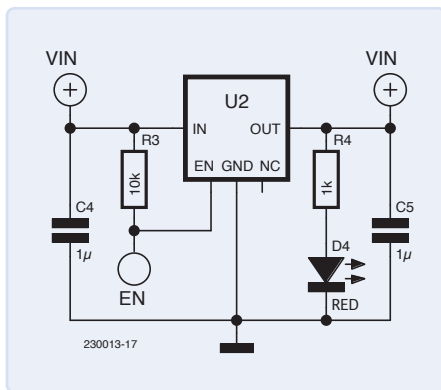


Figure 4: Artemis Voltage Regulator. ENABLE pin is pulled high by R3 (R1 on Artemis Nano). This resistor must be removed. The ENABLE pin is for the board's voltage regulator, not the MCU.

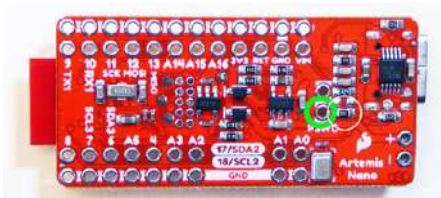


Figure 5: Artemis Nano board. R1 is circled in white. Unlike the Thing Plus, there is no EN connector but a "PSWC" connector. TPL5111 ENABLE should connect to the connector circled in green after R1 is removed. Note that R1 is on the back of the board, and it should be removed prior to mounting.

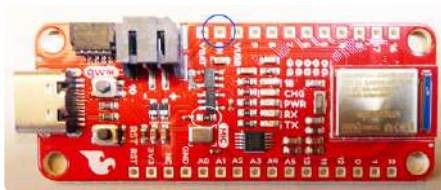


Figure 6: Artemis Thing Plus board. R3 is circled in white. TPL5111 ENABLE may be connected directly to the EN pin (circled in blue) after the resistor is removed.

These resistors set the voltage regulator ENABLE pin at "high" unless the pin is grounded. In the end, the modified boards were a nifty solution, with the entire circuit drawing only 20 μ A while asleep.

Both the TPL5110 and TPL5111 are interesting chips worthy of study, particularly in remote IoT sensing. The Adafruit boards are terrific implementations and provide excellent design flexibility.

Data Transmission

I chose HopeRF LoRa boards at 915 MHz for data transmission (other countries may require a different frequency). Multiple libraries are available for the Arduino environment that simplify implementation. The sub-gigahertz carrier frequency permits better penetration of walls and buildings and longer transmission distances. Transmission protocols are optimized for small data packets. The protocols do not require time-consuming handshakes and security overhead. However, this places the burden

of assuring data delivery on the programmer rather than on built-in transmission protocols.

At this point, you may ask how the sensor and display stations communicate. After all, the T_{ON} times of both devices overlap infrequently. If the sensor powers up and transmits data, the display station will likely be off. The solution was a third, always-on station based on a Raspberry Pi Pico containing no sensors but functioning as a data concentrator and forwarder, which is attached to a computer. I will refer to this as the home station. Its only peripheral is the radio and a few LEDs for debugging. Its voltage regulator can support a few hundred milliamperes of current, and I powered the radio from it. The Pico is powered by the USB port of a PC and communicates serially with a Processing program. Only a few hundred lines of code are enough to plot barometric pressure and serve up a table of readings on a web page. **Figure 7** illustrates the data flow as follows:

Data Flow Diagram

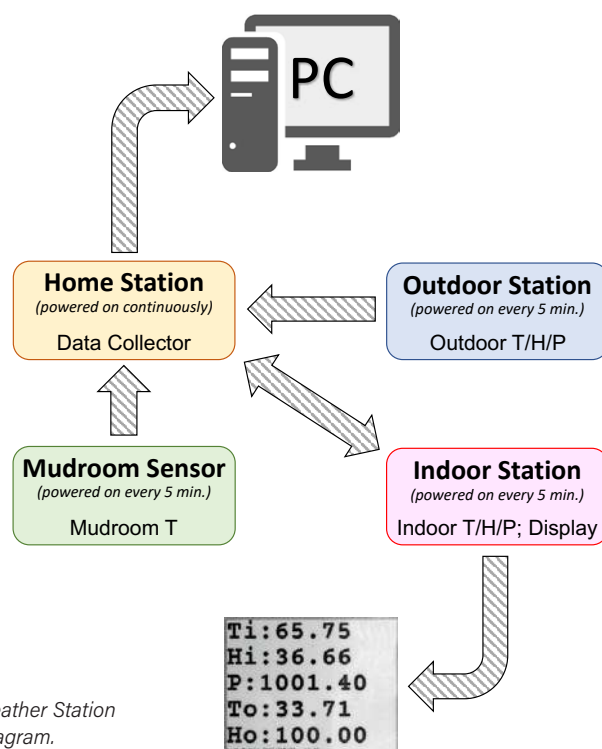


Figure 7: Weather Station data flow diagram.



Figure 8: Sensor board embedded in epoxy. Note the small plastic square in the upper-left-hand corner that prevented the epoxy from covering the sensor. Also note the USB-C right-angle dongle (left, middle) that permits access to the board for recharging and software updates.



Figure 9: Sensor housing detail. All vents are screened to prevent bio instead of software bugs ;-).

1. The outside sensor wakes every five minutes and sends temperature, humidity, and pressure (T/H/P) readings to the home station. There is no communication from the home station back to the outdoor sensor. The outside sensor shuts down after transmission without confirming receipt by the home station.
2. The home station stores the most recent readings from the outside sensor in RAM. The outside readings are also sent to the computer for further processing.
3. The inside sensor and display unit wakes up every 5 minutes. It measures indoor T/H/P, sends that information to the home station, and retains a copy of the measurements in RAM.
4. Upon receiving an update request from the indoor sensor and display, the home station posts this information to the computer. It then sends the most recent outside observations to the indoor station. After that, the indoor station updates the display and shuts down.

Over time, I noted repeated lockup of the home station with resultant failures to update the display and the Processing program. Postings on several support forums suggested that the LoRa Trans-

ceiver chip SX1276 could freeze and cause this behavior, locking up both the radio and the MCU when left running for extended periods of time. This behavior was never identified in the display or sensor stations because those modules had frequent power on/off cycles.

The solution for the Pico was to use a second MCU, an Arduino Mini, as a smart timer. Output from the Mini was connected to the enable pin on the Pico, shutting off the voltage regulator for 500 ms every hour. One output from the Pico was connected to the Mini, signaling when the data flow would be least vulnerable to the effects of a reset: The hard reset would occur only after there had been a data update and while the Pico was not receiving new information.

While this may seem to be an extravagant use of an MCU, the devices are so cheap that it is justified. I encourage input from the Elektor community on how to better solve this problem. Two versions of the home station software are provided [2], one for use with the Processing program, and one that is independent of a computer connection and requires only a power supply such as an old phone charger with the appropriate USB connector. The Processing code is a work in progress and is not presented as a complete, bug-free solution.

Weatherization

Initially, my outside sensor failed intermittently when there were rapid temperature and humidity changes, and I suspected condensation. After examining all my solder joints and wiring and finding no faults, I decided to pot my outside board. I 3D-printed a container for my circuit board and a small dam to place around the sensor board so it would not be submerged in epoxy (**Figure 8**). I connected a 90° USB-C dongle, which allowed me to retain access to the Artemis board's USB-C connector.

I used inexpensive clear craft epoxy, available on Amazon. Craft epoxy is formulated to cure very slowly and gives the user an extended window to work with the material. Since I was not seeking a perfect aesthetic outcome and didn't want my idiot cats covered in epoxy, I put the container on an electric food warmer at about 150°F (65...66°C). The epoxy was paw-proof within an hour. I then allowed curing to finish overnight at room temperature (18–20°C). The circuit board is housed within a ventilated plastic jar painted white (**Figure 9**) and the ventilation holes are screened to bug-proof the sensor.

Before I potted the circuit, I confirmed, through testing, that the epoxy would neither short-circuit bare wires nor disrupt



Component List

Resistors

R1 = see text
 R2..R4 = 1 M Ω ; 1/4 W
 R5 = 680 Ω ; 1/4 W
 R6..R8 = 360 Ω ; 1/4 W

Semiconductors

D1 = LED blue
 D2 = LED red
 D3 = LED yellow
 D4 = LED green

Boards

2 \times TPL5111 breakout boards (Adafruit Industries)
 1 \times Raspberry Pi Pico
 1 \times Artemis Thing Plus (Sparkfun)
 1 \times Artemis Nano (Sparkfun)
 2 \times BME280 breakout boards (ASIN B08DHTGNHR *)
 1 \times FTDI basic breakout 3.3 V (Sparkfun)
 3 \times HopeRF RFM95CW 915 MHz FCC Certified LoRa Transceiver (Anarduino.com)
 1 \times Arduino Pro Mini, 3.3 V 8 MHz (see text)

Other

2 \times ST 4 pin Connector (ASIN B01DUC1M2O*; see Building Notes)
 1 \times 4.2" ePaper Display (WaveShare; ASIN B074NR1SW2*)
 3 \times double sided FR4 perf boards 7 X 9 cm (ASIN B08F7X8JHV*)
 2 \times 18560 LiPo rechargeable batteries
 2 \times battery holders for 18650 batteries
 Epoxy Resin kit (ASIN B07TVWTG829*)
 USB-C 90° adapter (ASIN B0BBVWF54L*)
 Headers and wires as needed

* ASIN codes refer to searchable Amazon inventory numbers.

electrical connections. Since I potted the circuit, there have been no further intermittent failures.

Software and Data

The software for all three stations uses LoRa libraries and WaveShare code. For the indoor and outdoor station, libraries for the BME280 sensor chip are necessary. You can download the complete packages at [2] for free. The coding burden for the TPL5111 is trivial. I traded code simplicity and shortened T_{ON} in exchange for guaranteed data delivery. After measurement, a short string is created that contains the observations and sending station identification; this string is sent twice (see below). Transmissions from the home station to the display station also include time data so that the display can tell the time of the last update.

The code may appear incomplete. However, code execution halts when the voltage regulator is turned off. To achieve this, the MCU sends a *DONE* signal to the timer. This closes any gaps in the software. Except for the WaveShare code, I have not included libraries in the software download. These are easily added to your environment by the Arduino IDE. I used the Sparkfun BME280 library because there are no code dependencies other than *Wire.h*. The Sandeep Mistry LoRa library provided all necessary functionality and had no code dependencies other than *SPI.h*.

Since the home station cannot coordinate transmissions among the sensors and the display station, data collisions with resulting data loss are inevitable. The data string is sent twice with a delay between transmissions to reduce the risk of undelivered data.

In the final analysis, if a few data points per day are lost, this is acceptable for a non-critical application.

During development, I realized that the home station could communicate with a computer and send the weather observations to a database, a web page, or similar. This is a work in progress. I have written a crude web page server in Processing, a Java derivative language with an extended user base and many good libraries. As said, the Processing code is provided, as well as the Arduino sketches. Although there may be some superficial similarities, this is not a LoRaWan project, and the home station is not a LoRa gateway.

My configuration monitors outdoor conditions, the conditions in my kitchen (location of the display), and a third sensor in our mudroom, which is at freezing risk in winter. The mudroom sensor data appears only on the webpage. There are upper limits to the number of sensors and displays that can be supported with this arrangement. As network traffic increases, data loss due to collisions will eventually become unacceptable. Many factors will affect this, including the accuracy of the timing resistors for the TPL5111, environmental factors, and, most importantly, sampling and/or update request frequency. There is no upper limit on the number of displays not making data requests. Units making data requests contribute to radio traffic. The total radio traffic, rather than sensor measurement transmissions, limits the network. Calculating the theoretical limits of various configurations is left as an exercise for the reader.

Practical Considerations

The hardware is hand-wired. Read the **Building Notes** frame carefully. The sensors communicate via I²C. I used the Sparkfun Qwiic system, which is very convenient. It is easy to wire a Qwiic-compatible connector to the small BME280 boards available on Amazon and eBay. Be careful to purchase a **BME** rather than **BMP**280 if you want humidity information. The HopeRF module has a 2 mm, rather than a 2.54 mm pitch, requiring a breakout board or careful soldering. I am using a simple 1/4-wave wire antenna.

Building Notes

1. **Removing R1 and R3:** See pictures for identification of R1 and R3 resistors that need to be removed from Nano and Thing respectively. If you do not, the circuits will not work.
2. **ENABLE connection on Artemis Nano:** See Figure 5 for ENABLE connection on Nano. I took a picture of the bottom of the board. You should NOT use the pin at ground potential!
3. **Main circuit board:** Any perf board can be used, but the referenced items were pre-tinned, FR4, adequate size, and stood up to rework.
4. **Radio frequency:** Purchase radio board legal for your country.
5. **Connecting to the Qwiic system:** The JST connector mentioned fits perfectly into the QWIIC system, but colors DO NOT correspond to Sparkfun convention. For these connectors, white = GND, yellow = 3.3 V, black = SDA, and red = SCL.
6. **BME280 boards:** Be careful purchasing BME boards. Some Amazon vendors substitute (knowingly or not) BMP280 for BME280 chips. The BMP version is less expensive and looks almost exactly the same, but the libraries for BME won't work and you won't get humidity data.
7. **WaveShare connection notes:** WaveShare unit comes with an easy to use cable. Purple = BUSY, White = RESET, Green = DC, Orange = CS, Yellow = CLK, Blue = DIN (MOSI), Brown = GND, Grey = Vcc. The unit does not have a MISO connection.
8. **Time interval of the TPL5111:** The TPL5111 timer interval is programmed using either an on-board potentiometer or external resistor. The Adafruit website and the IC datasheet provide a table of resistor values for various time intervals. If you use an external resistor, you need to cut a trace on the back of the board. If power is a real issue, the activity LED can also be taken out of the circuit by cutting a trace.
9. **Connecting the battery to Artemis boards:** Battery connects to the board with a widely available 2 wire JST cable.
10. **Radio breakout board:** I used a breakout board for the radio from Diycon.nl (LoRa Node PCB 100 Shield Only for RFM92/RFM95). There it is easy to attach either a wire antenna or an antenna connector.
11. **Radio antenna:** I used a simple $\frac{1}{4}$ wave wire antenna. The correct wire length for 915 MHz is 78 mm, soldered to the middle connector on the radio breakout board.
12. **Epoxy:** Very messy. Gloves, disposable containers for measuring and mixing, disposable mixing sticks, and drop cloth all needed. Make sure the board is working perfectly before proceeding. The 90° USB-C adapter must be installed, and the timer should have the right resistor. The shallow container must be slightly larger than the board and battery. Don't submerge the BME280 in epoxy. I made a dam around the sensor (epoxy will come up through the perf holes, so protect the bottom too). Alternatively connect the sensor via wires to the QWIIC connector and hold the sensor board above the epoxy while it cures. Gently heating the epoxy accelerates curing time considerably. The three key endpoints are that the sensor is not covered with epoxy, the USB-C connector remains accessible through the adapter, and that with the exception of the sensor board, all the components on the circuit board are covered.
13. **The housing:** There are several design issues. Holes for ventilation should be located and protected such that for water to infiltrate into the inside of the housing, it would need to flow up against gravity. Second, the housing should be reflective or at minimum painted white to minimize greenhouse effect. Third, the housing should be lightweight. The thermal inertia of a large, heavy housing will make your measurements slow and inaccurate. Finally, the case should be insect-proofed in some manner.
14. **18560 batteries:** Batteries boasting impossible energy densities at ridiculously low prices are ubiquitous on Amazon and eBay. Be skeptical of these claims. Branded cells with 2,500...3,500 mAh are a safe choice!

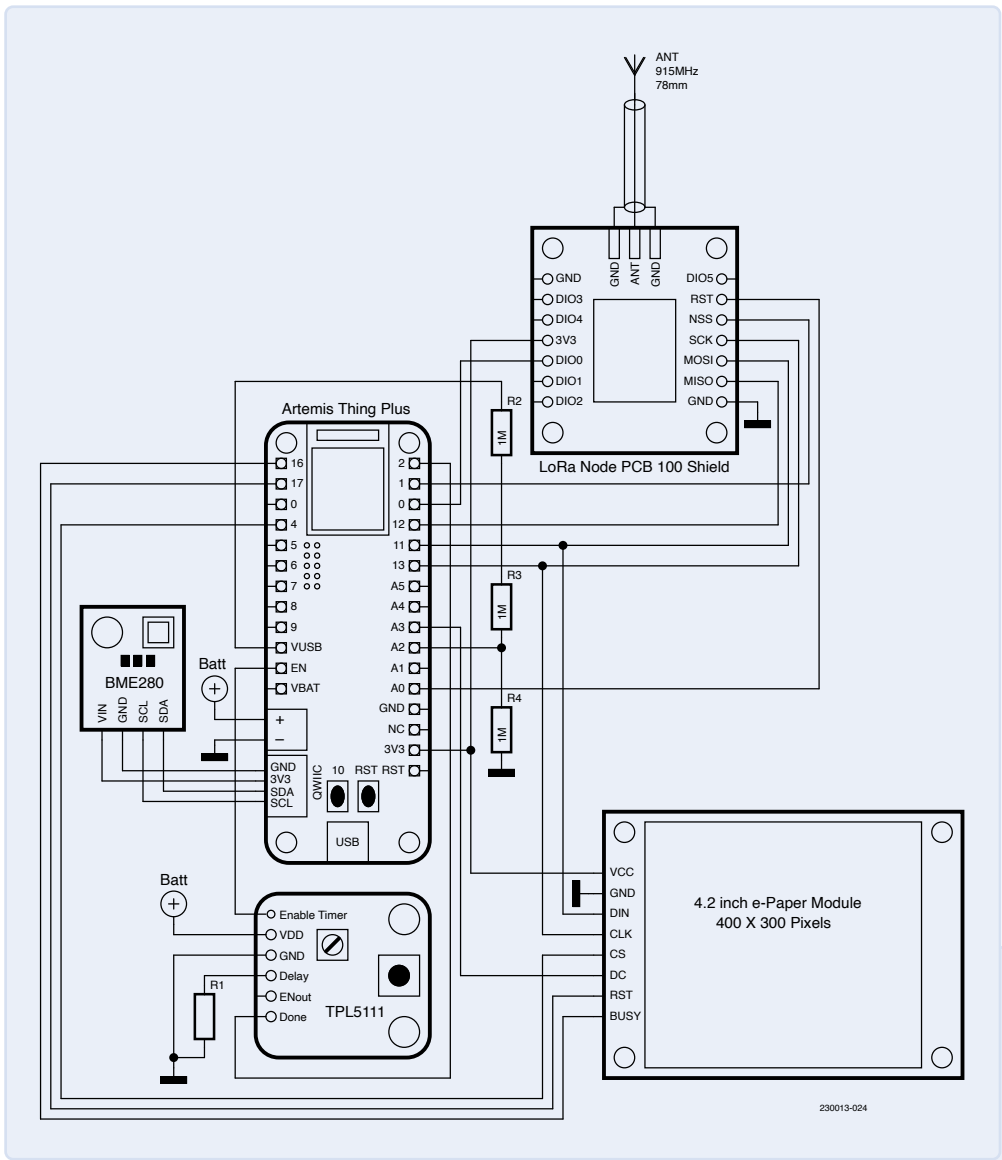


Figure 10: Circuit of the Display Station.

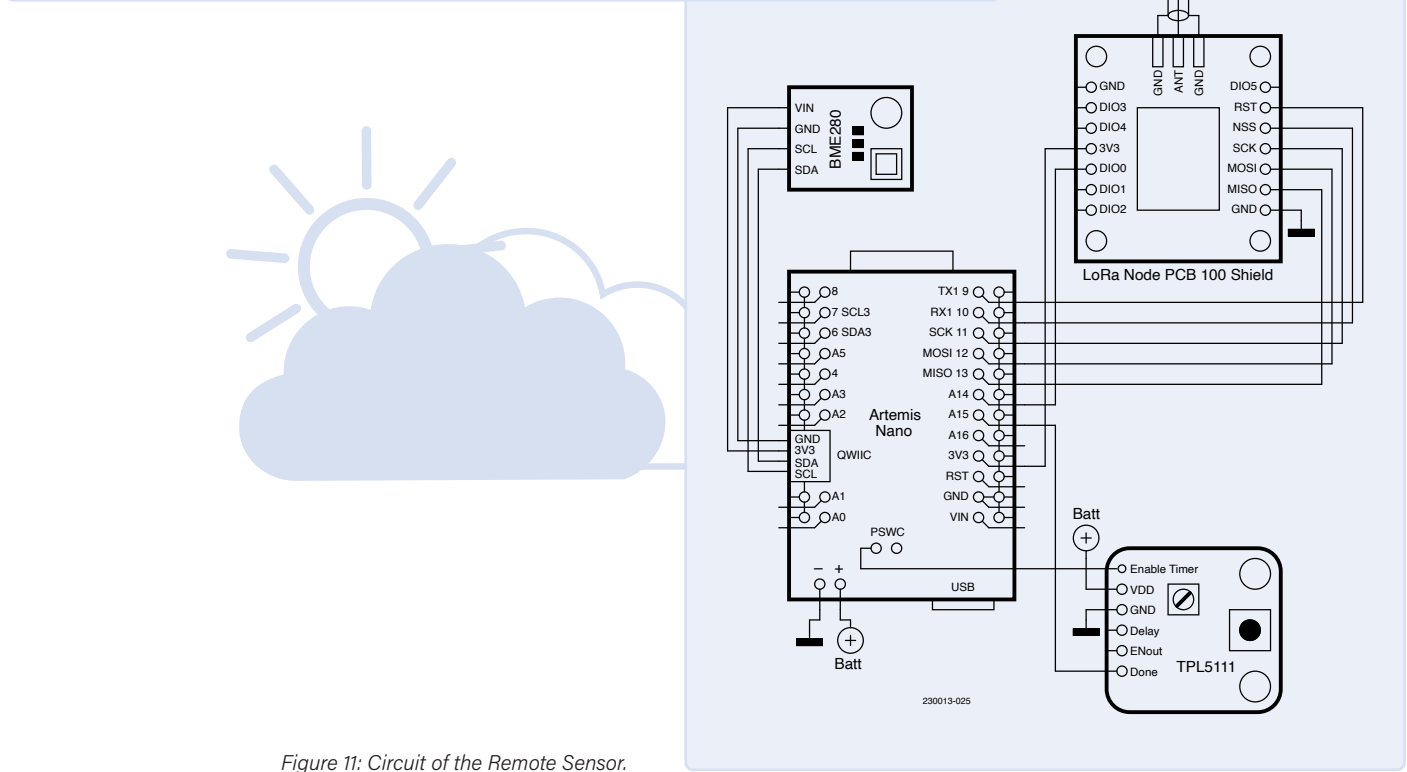


Figure 11: Circuit of the Remote Sensor.



I used FontEdit to create a 36-point font for the ePaper display. WaveShare's instructions are hard to understand, but ultimately I developed working code. If you use an MCU other than the Artemis, be mindful that the font tables and bitmap buffer consume a lot of RAM.

I used a rechargeable Lithium 18560 battery for power. See the Building Notes for observations on sourcing 18560s. Other power configurations are possible, but alkaline batteries are emphatically not recommended for subzero (0°F, -18°C) temperatures.

The schematics (see **Figures 10 to 12**) are a suggestion. The popular new 32-bit boards are flexible, with many pins supporting interrupts and with alternative SPI, I²C, and UART interfaces. As such, some connections were based on optimizing the overall circuit's physical layout.

Enjoy! 

230013-01

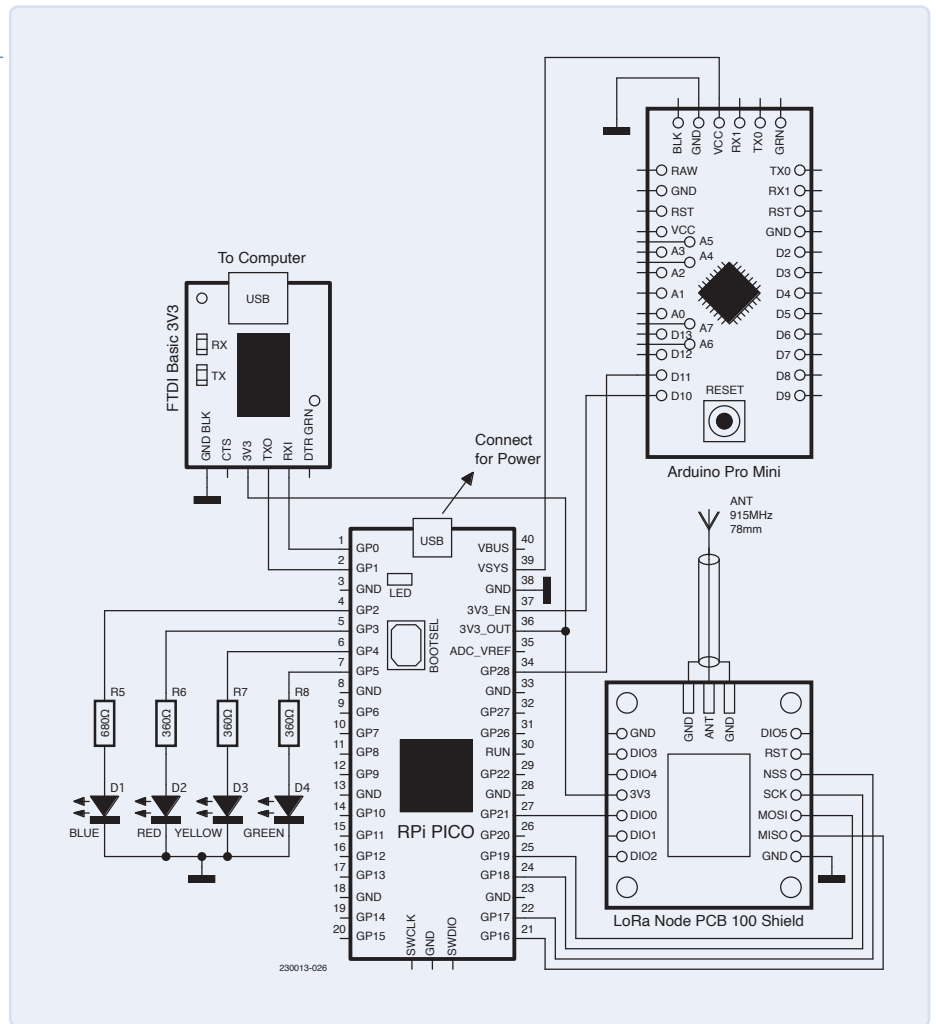


Figure 12: Circuit of the Home Station.

About the Author

Ed Ringel is a semi-retired respiratory and critical care physician. He enjoys the Maine outdoors with his wife, writes science fiction, makes things with his 3D printer, and develops electronics projects with MCUs.

Questions or Comments?

Do you have technical questions or comments about this article? Email Elektor at editor@elektor.com.



Related Products

- **SparkFun Sensor Kit**
<https://elektor.com/19620>
- **Raspberry Pi Pico RP2040**
<https://elektor.com/19562>



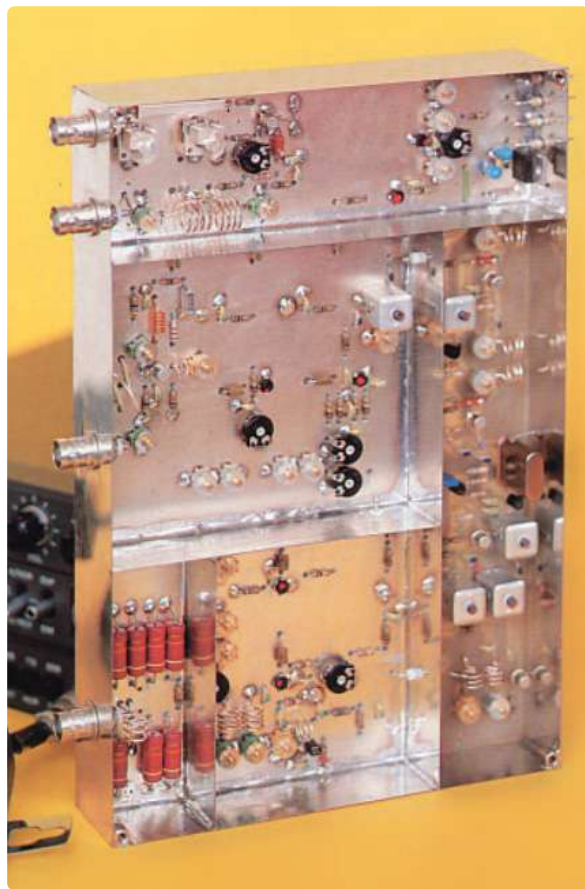
WEB LINKS

- [1] Unsuccessful adventures: <https://forums.adafruit.com/viewtopic.php?p=927526>
- [2] This project on Elektor Labs: <https://elektormagazine.com/labs/low-power-lora-weather-station>

Transverter for the 70 cm Band

By Jan Buiting PE1CSI (Elektor)

In 1981, Elektor wooed the ham radio community with a beautifully engineered transmitter/receiver converter (transverter) for the 70 cm band, then a pretty vacant stretch of the radio spectrum used by real experimenters to communicate over the ether without cellphones, can you imagine?



Prototype of the 70 cm transverter built by Gerrit, PA0HKD and Ed, PE1CJP for Elektor Labs in 1981. That's an Icom IC211 2 m all-mode transceiver driving the transverter.

The 70 cm (430–440 MHz) transverter I'm reminiscing about here was published in two parts in *Elektor*, June and October 1981 [1][2]. It is a fine example of a publication aimed at radio amateurs not willing to pay the exorbitant prices, at that time, of commercial equipment. They simply wanted SSB (single-side-band) on 70 cm, the way they had been able to enjoy it on shortwave as well as on 2 m (144–146 MHz) for many years. As opposed to FM, SSB is a linear mode requiring perfect linearity of all transmitter stages right up to the antenna connector.

Caution: Hams at Work

The original design for the transverter came from J. de Winter PE0PJW. At Elektor Labs, staffer Gerrit Dam PA0HKD and lab apprentice Ed Warnier (PE1CJP, now PA1AW) tweaked and Elektorized the design to the extent of being reproducible by Elektor readers as well

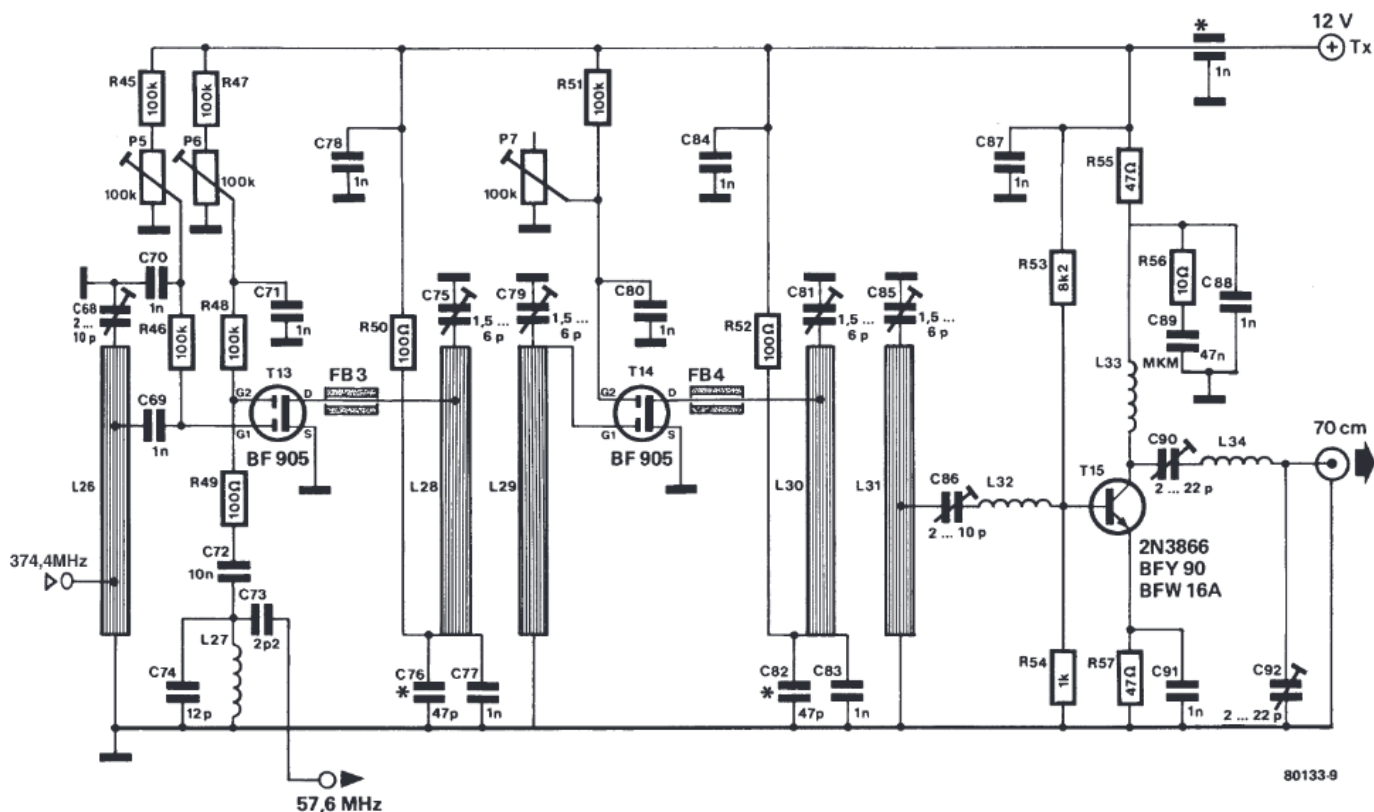
as being legally compliant in terms of spurious emissions and harmonics.

Ed remembered that putting an ultra-high-frequency (UHF) design on a PCB was "a challenge," having to struggle with Elektor's PCB designers quite unused to the vagaries of sub-pF stray capacitances and nanohenry coils, and were only comfortable with "DC stuff" like audio, microcontrollers, and PSUs. In the end, a fortuitous solution was found in the use of microstrip line inductors etched

on the board for the UHF portions of the circuit, not forgetting a silver-plated PCB and, of course, tin-plated sheet metal "walls" on the board to keep spurious radiation to a minimum.

Loud & Clear

During the early 1980s, the 70 cm band had a particular attraction, not just as a meeting place for hams with 100% home-built rigs (including amateur television, ATV), but also for satellite communication, which enabled

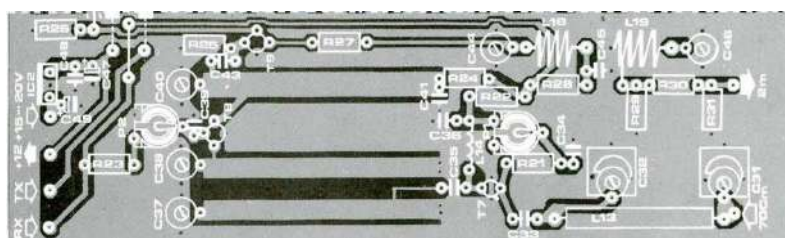


Circuit diagram of the transverter's UHF section only. In 1981, those "microstrip line inductors" required a new symbol to be created by the erstwhile Elektor drawing department, as well as one word added to the editors' technical dictionary.

cross-continental QSOs in CW and SSB, using relatively low transmit powers and highly directional antennas.

Elektor's all-DIY 70 cm transverter reached a high Q factor among hams, arriving well-timed and tuned spot-on to their requirements. The project was brilliantly engineered and gloriously documented in two magazine articles, resulting in a UFB (*ultra-fine business*) publication. ◀

220214-01



A section of the transverter circuit board, reproduced from the October 1981 issue. Those rectangular areas between thick black lines aren't big fat short-circuits, but tuned inductors "microstrip" style, operating at around 430 MHz. They aren't something you can drag, drop, and autoroute!

WEB LINKS

- [1] J. de Winter, "Transverter for the 70 cm band (1)," Elektor 6/1981:
<https://www.elektormagazine.com/magazine/elektor-198106/44626>
- [2] J. de Winter, "70 cm transverter (2)," Elektor 10/1981:
<https://www.elektormagazine.com/magazine/elektor-198110/44750>

Questions or Comments?

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

Climate Calling Engineers

Move Fast and Fix Things

By Priscilla Haring-Kuipers

The sixth synthesis report by the Intergovernmental Panel on Climate Change (IPCC) is clear: “Human activities, principally through emissions of greenhouse gases, have unequivocally caused global warming.” What are we going to do about it?

The sixth synthesis report by the Intergovernmental Panel on Climate Change (IPCC) is the collective scientific wisdom on climate change and how to fix it. [1] They inform the UN. The main message of their latest report has hope: “There are multiple, feasible and effective options to reduce greenhouse gas emissions and adapt to human-caused climate change, and they are available now,” [2] but currently, we are not applying the technical solutions we have with enough vigour, scale or speed.

Calling all engineers! There is a lot to do, so let me give you some highlights:

Current Climate

We are now at 1.1°C global warming and will likely reach 1.5°C in the early 2030s and shoot up to 3.5°C this century if we don’t change drastically. “There is a rapidly closing window of opportunity to secure a liveable and sustainable future for all.” [1] We have already caused a lot of damage across ecosystems. More than climate scientists estimated earlier. We have lost many species, nearly 50% of coastal wetlands, and we are impacting ecosystems in ways that are not reversible. Cities have become hotter and the air we breathe more polluted.

Our supply chain has already been impacted by the more frequent occurring extreme weather, making factories freeze or catch fire. Water is fast becoming a contested resource, and factories should look into either recycling or using seawater.

We have not done nothing. Agreements made at Kyoto and Paris have helped. Social movements have accelerated climate action. We can still save ourselves with climate resilient development based on science, indigenous knowledge and local context. High-tech and low-tech solutions working together.

“Individuals with high socio-economic status contribute disproportionately to emissions, and have the highest potential for emissions reductions, e.g., as citizens, investors, consumers, role models, and professionals.” [1] That means us. What we do and what we demand of our governance makes a big difference. What you choose to work on as an engineer will either contribute to a liveable world or to further heating up the place. When we support developing regions with our technological development, they can leapfrog to low-emissions solutions with us.

Stop That

If we are ever to stay under 2°C of global warming, a lot of fossil fuels are going to have to stay in the ground. Today, new fossil fuel developments are still being funded, and the fossil fuel industry receives more money in private investments, public subsidies and tax breaks than developments tackling climate adaptation and mitigation. [1] Simply ending fossil fuel subsidies would lower greenhouse gas emissions with 10% by 2030, while improving public revenue that could be redirected to our necessary transition. If your work or your pension funds are connected to the fossil fuel industry, you might want to start looking for a way to decouple before the well is shut down. Our electronics are heavy on petrochemicals and will be looking to shift to bio-based alternatives. Opportunities abound.





By Priscilla Haring-Kuipers, made with DALL-E: Electronics engineer soldering a product to achieve a future save from climate change.

Carbon pricing such as carbon taxes or emission trading have led to some low-cost emission reduction measures but have not been very successful to promote the higher-cost measures that are necessary to shift an industry. We need more. Luckily, climate laws are increasing, and they are helping to fight climate change causes and effects. Climate-related litigation is growing and has already had an effect on the “outcome and ambition of climate governance.” [1] It is likely that climate law will grow in the near future, both internationally and on regional levels. The WEEE regulations and the Supply Chain Act are early versions of what is coming. Your efforts and your company can be ahead of the curve, riding the green wave, or you can be dragged along by legislation, but everyone is coming eventually.

Technology for the Win

“If all technically available options were used, global emissions could be at least halved by 2030, at manageable costs.” [1] We need tons of engineers to roll out, scale up, improve and adapt to local circumstances many of the already available and proven solutions.

In the last decade, the cost of solar energy has dropped by 85%, wind energy by 55% and lithium-ion batteries by 85%. Meanwhile, deployment has increased over tenfold for solar and over a hundred-fold for electric vehicles. In some areas and industries, keeping the old is becoming more expensive than changing to the new. Work on whatever you can to push, develop and spread this development.

Green energy will not only curb our emission, but the economic benefit in air quality alone would offset the cost of the transition. Co-development of energy efficiency and renewable energy will create a happy feedback loop of improvement. Work on big renewables and small-scale nets, smart-grids, transmission and capacity is very much needed.

Cities are critical in this transition. We can build or retrofit to match our new low-emission lifestyles and make space for cycling and walking, teleworking and electric public transport. More plants and water in cities would help cool during heatwaves, process heavy rainfall slower and keep moist during droughts while benefiting the health and well-being of all who live there. Engineers should work on building materials and practices, sustainable urban planning and maintenance, digital communities and smart transport solutions. Many cities have already announced a net-zero emissions target. My city started a green jobs market for all the technical roles we are going to need to develop, install and maintain this bright new future. Your city might have a similar initiative.

Carbon capture is most reliably done by reforestation, improved forest management, soil carbon sequestration, peatland restoration and coastal blue carbon management. Protecting high-carbon ecosystems would have an immediate impact. Globally, we need to protect 30-50% of our land and water to maintain a resilient biodiversity. Throw your skill set behind any project that supports conservation and restoration.

Your time is now. ◀

230265-01

No Geo-Engineering

Blocking the sun with solar shields or sulfur is a no-go. Short-term and local cooling effects are likely, but the amount of green house gases would still grow and the acidification of our oceans would continue. We don't know enough about the effects on the targeted region nor our global ecology. Once it is up and running, turning it off could cause “rapid climate change.” The risks are too great, and the reward is too uncertain.

WEB LINKS

- [1] IPCC, “AR6 Synthesis Report: Climate Change 2023,” 2023: <https://www.ipcc.ch/report/ar6/syr/>
- [2] IPCC, “Urgent climate action can secure a liveable future for all,” March 20, 2023: <https://www.ipcc.ch/2023/03/20/press-release-ar6-synthesis-report/>

The Elektor Store

Never expensive, always surprising

The Elektor Store has 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

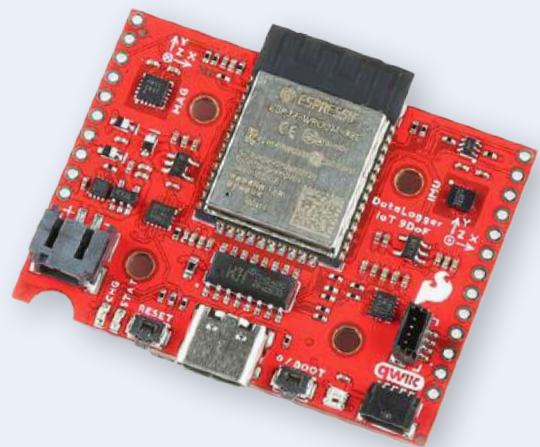
for surprising electronics. We offer the products that we ourselves are enthusiastic about or that we simply want to try out. If you have a nice suggestion, we are here: sale@elektor.com.

SparkFun DataLogger IoT (9DoF)

The SparkFun DataLogger IoT (9DoF) is a data logger that comes preprogrammed to automatically log IMU, GPS, and various pressure, humidity, and distance sensors. All without writing a single line of code! The DataLogger automatically detects, configures, and logs Qwiic sensors.

Price: €94.95

Member Price: €85.46



www.elektor.com/20487

Miniware MDP-XP Digital Power Supply Set (MDP-M01 + MDP-P906)



MDP (Mini Digital Power System) is a system of programmable linear DC power supply based on modular design, capable of connecting different modules for use as needed. MDP-XP consists of a display control module (MDP-M01) and a digital power module (MDP-P906).

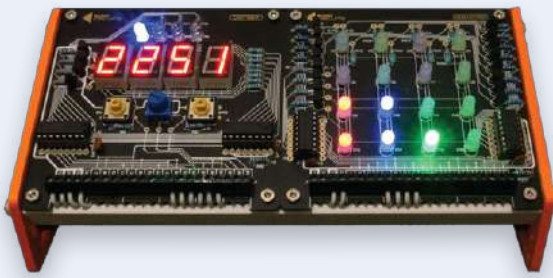
Price: €269.00

Member Price: €242.10

www.elektor.com/20458



Short Circuits: The 4-Pack (Arduino-compatible Electronics Platform)

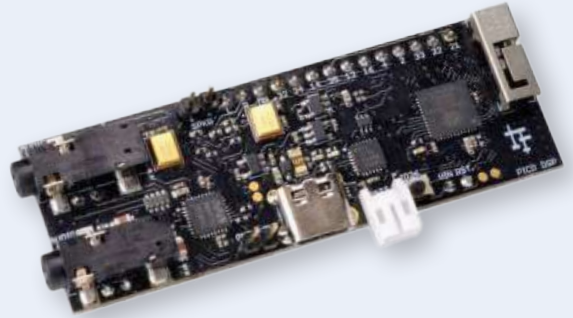


Price: €99.95

Member Price: €89.96

www.elektor.com/20474

PÚCA DSP ESP32 Development Board

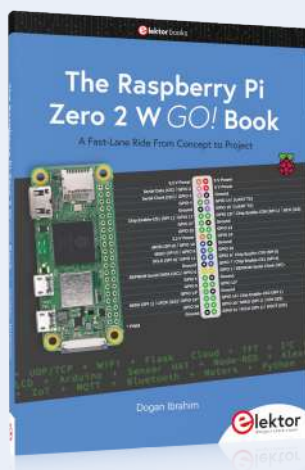


Price: €69.95

Member Price: €62.96

www.elektor.com/20504

The Raspberry Pi Zero 2 W GO! Book



Price: €34.95

Member Price: €31.46

www.elektor.com/20445

The Elektor Power Supply Collection (USB Stick)



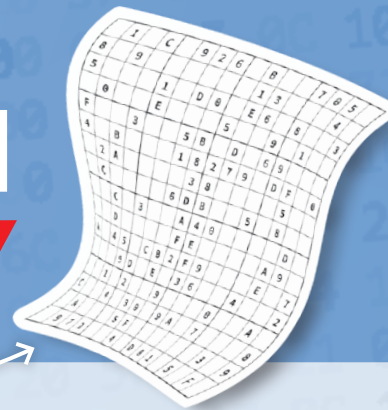
Price: ~~€49.95~~

Special Price: €34.95

www.elektor.com/20451

Hexadoku

Puzzles with an Electronic Touch



Traditionally, the last page of *Elektor Magazine* is reserved for our puzzle with an electronics slant: Welcome to Hexadoku! Find the solution in the gray boxes, submit it to us by email, and you automatically enter the prize draw for one of five Elektor Store vouchers.

The Hexadoku puzzle employs digits in the hexadecimal range 0 through F. In the diagram composed of 16×16 boxes, enter digits such that **all** hexadecimal digits (that's 0–9 and A–F) occur once only in each row, once in each column, and in each of the 4×4 boxes (marked by the thicker black lines). A number of clues are given in the puzzle, and these determine the starting situation.

Correct entries received enter a prize draw. All you need to do is send us **the digits in the gray boxes**.



SOLVE HEXADOKU AND WIN!

Correct solutions received from the entire Elektor readership automatically enter a prize draw for five Elektor store vouchers worth **€50 each**, which should encourage all Elektor readers to participate.

PARTICIPATE!

By August 15th, 2023, supply your name, street address and the solution (the digits in the gray boxes) by email to: hexadoku@elektor.com

PRIZE WINNERS

The solution of the Hexadoku in edition 5-6/2023 (May & June) is: **EB0C8**.

Solutions submitted to us before June 15th were entered in a prize draw for 5 Elektor Store Vouchers.

The winners are posted at elektormagazine.com/hexadoku.

Congratulations, everyone!

D		1	6	0		2	7		A						B
	3								0	1	8	2	7		
5		F					4	3	B		D			0	A
A					3	5					F			4	
3				4			1	9	C				E		
			1		B			6						C	8
6			5			9	0	A	8					D	F
0		A		C		D				1		4	5	6	7
	D	6		8	4	F			E			2			
2	E	8		5		0		7	4			6	1		
	F						6			2	B			5	
	5	0	4							9		C		E	8
	6			9			3	C	7		0				
					5	4	2		9						
		5	2		0	E	A			D	8			B	
8		9					F				4				6

E	2	5	A	0	3	7	B	1	F	8	C	6	4	9	D
6	C	D	F	1	5	8	9	3	2	4	0	E	B	A	7
B	0	1	7	6	E	4	F	D	9	5	A	C	8	3	2
8	3	4	9	2	A	C	D	6	7	B	E	F	0	1	5
D	5	F	8	4	B	A	7	C	1	6	3	0	E	2	9
7	4	2	3	E	0	1	C	F	5	9	B	D	6	8	A
9	6	B	0	3	D	F	8	2	A	E	7	5	1	4	C
A	1	C	E	5	2	9	6	4	8	0	D	7	3	B	F
C	7	8	2	A	9	D	3	0	E	1	6	B	F	5	4
F	D	A	1	7	6	0	2	5	B	3	4	8	9	C	E
3	E	9	6	B	8	5	4	A	C	7	F	1	2	D	0
4	B	0	5	C	F	E	1	8	D	2	9	3	A	7	6
0	9	E	B	D	1	2	A	7	3	F	5	4	C	6	8
1	A	7	D	9	4	6	E	B	0	C	8	2	5	F	3
2	8	6	C	F	7	3	5	E	4	A	1	9	D	0	B
5	F	3	4	8	C	B	0	9	6	D	2	A	7	E	1

The competition is not open to employees of Elektor International Media, its subsidiaries, licensees and/or associated publishing houses.



PROTEUS DESIGN SUITE

Design Quality Assurance

Constraint Driven Design

Flexible and scalable
rule system

Full support for design
rule rooms

Manufacturing
solder mask rules

Live display of
violation areas

Zone Inspector

Analyze plane coverage and
stitching

Grid view of plane
configurations

Edit plane settings and
draw order

Pre-Production Checklist

Set of board tests
before Gerber Output

Includes placement,
connectivity and
clearance testing

Completely independent
code for clearance checks

Dedicated Reporting Module

Tables automatically
populate with design
data

Compliance status for
diff pairs and length
matched routes

Make custom
reports with data
object tables

Generate reports
from templates



Widest selection of electronic components™

In stock and ready to ship



[mouser.co.uk](https://www.mouser.co.uk)

