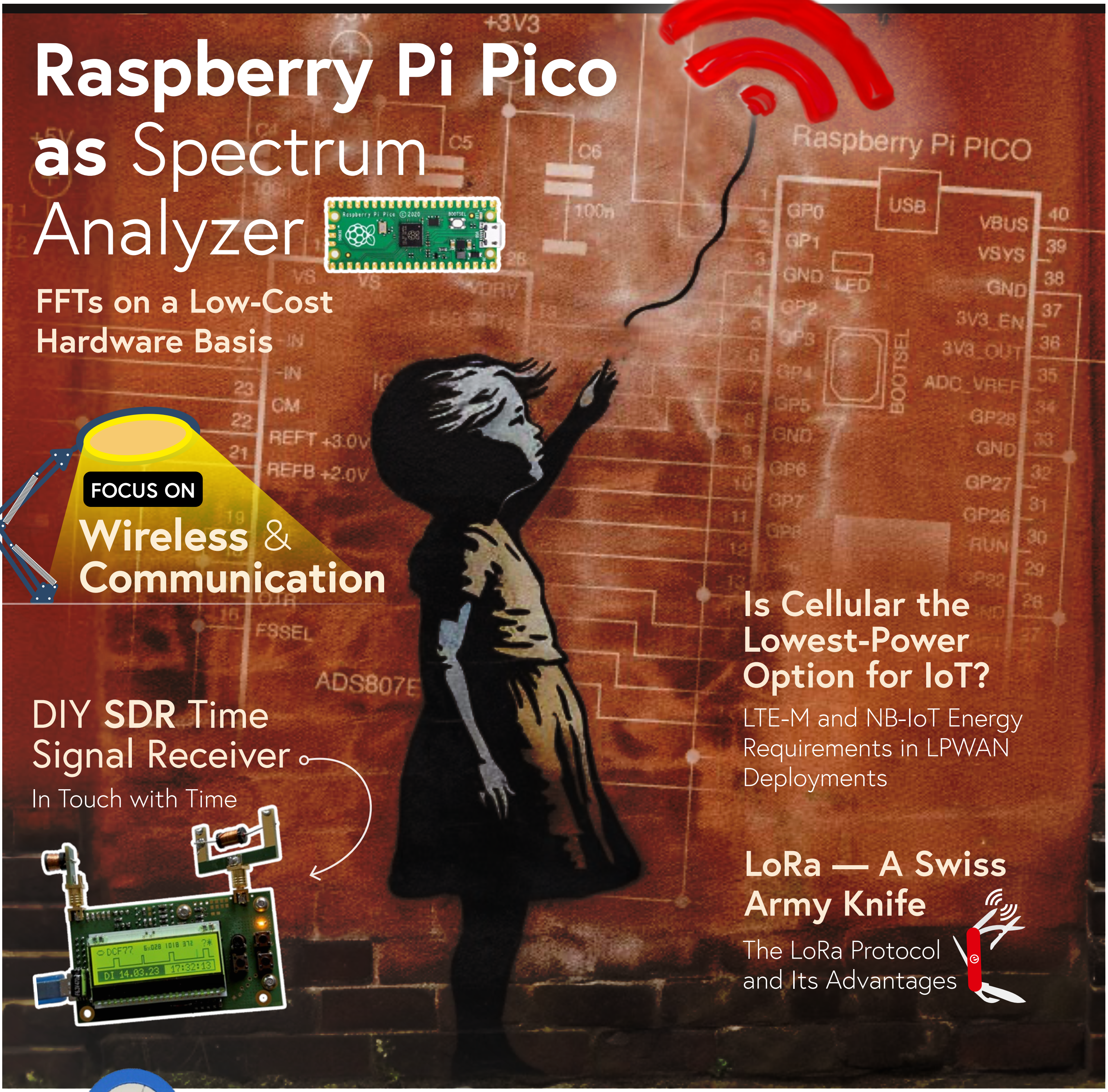
£9.95 | €10.95 | $12.95

elektor MAG

design > share > earn

524 · SINCE 1961

SEPTEMBER & OCTOBER 2023
ELEKTORMAGAZINE.COM



Raspberry Pi Pico as Spectrum Analyzer

FFTs on a Low-Cost Hardware Basis

FOCUS ON
Wireless & Communication

DIY SDR Time Signal Receiver

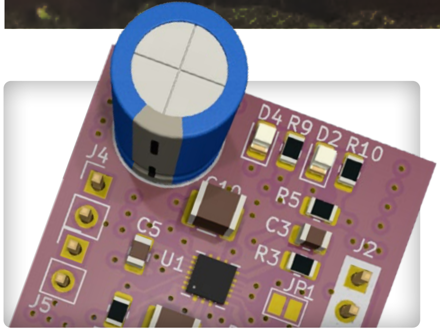
In Touch with Time

Is Cellular the Lowest-Power Option for IoT?

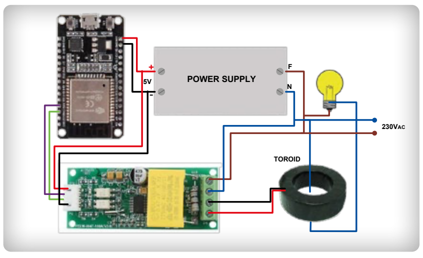
LTE-M and NB-IoT Energy Requirements in LPWAN Deployments

LoRa — A Swiss Army Knife

The LoRa Protocol and Its Advantages



Motor Driver Breakout Board
5 A DC Motor Driver at Just 3×3 mm
p. 56



Cloud-Based Energy Meter
With ESP32 and PZEM-004T Voltage/Current Sensor
p. 102

±40 V Linear Voltage Regulator Power for the Fortissimo ... and Others!
p. 18

Electronic Components

TRANSFER
MULTISORT
ELEKTRONIK

GLOBAL DISTRIBUTOR OF ELECTRONIC COMPONENTS

Enjoy ECAD Model at TME

PCB footprints & symbols

3D models

Quick access

A perfect solution for design engineers

Transfer Multisort Elektronik Sp. z o.o.
Łódź, Poland, export@tme.eu

Join us:

tme.eu

YOU NEED IT, WE HAVE IT!

tme.com

Volume 49, No. 524
September & October 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

Cover Design
Midjourney, created by Harmen Heida

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

From Simple to Modern

Our future will largely be wireless: At present, various companies and universities are already researching 6G, which is expected to achieve data rates of up to 400 Gb/s and operate at frequencies up to 3 THz, and all this is aimed to start in around 2030. The spectrum of wireless applications is already vast today. It ranges from position-determination to energy-saving long-distance data transmission to using a smartphone as a remote control. In any case, we had so many articles and ideas for this magazine issue that it was impossible to fit everything into one edition. The diversity of electronic topics, for which we are particularly appreciated by our varied-interest readers, would otherwise have been compromised. Some of our wireless articles — including an interesting report on centimeter-accurate position determination at maker-friendly costs — can be found in the upcoming issues.

And for those particularly interested in specific areas such as "Power Electronics," "Wireless," "Measurement and Testing," or "Electronics Production," we will soon be offering special theme pages on our website, elektormagazine.com. There, you will find not only articles from our magazines but also exclusive online content. Background reports and (video) tutorials for beginners are complemented by additional projects from our global community.

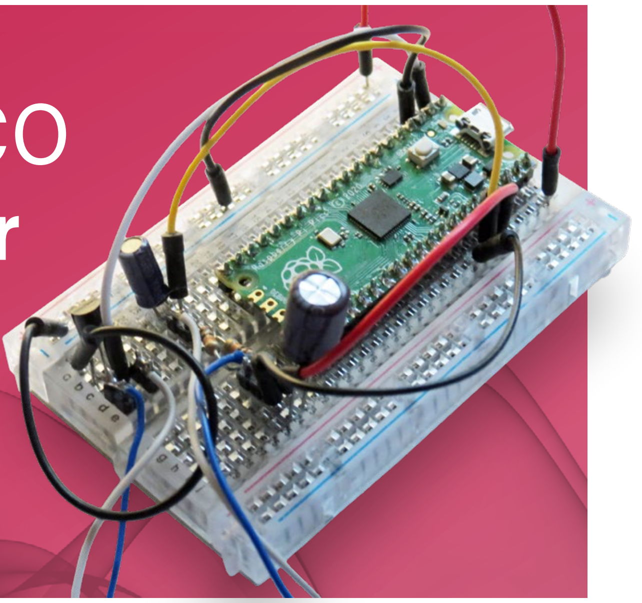
Perhaps I will also introduce my very personal wireless project there, a wirelessly controlled speaker system for outdoor use, powered by individual lithium batteries. Although everything is (still) composed of purchased components, considerable development work has already gone into it. Now, you might want to know which hypermodern wireless standard I use? Well, it's good old FM radio, but at 863...865 MHz, which is also used at "Silent Disco" parties. When it comes to straightforward setup and the lowest latencies, simplicity is still key!

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, Saad Imtiaz, Alina Neacsu, Dr. Thomas Scherer, Clemens Valens, Brian Tristam Williams | **Regular Contributors:** David Ashton, Tam Hanna, Ilse Joostens, Prof. Dr. Martin Ossmann, Alfred Rosenkränzer | **Graphic Design & Prepress:** Harmen Heida, Sylvia Sopamena, Patrick Wielders | **Publisher:** Erik Jansen | **Technical questions:** editor@elektor.com

THIS EDITION

Volume 49, No. 524
September & October 2023



MCU Wireless Communication Made Flexible
EEPROM Opens Networking Prospects for Wireless MCUs
24

Projects

Next Edition

Elektor Magazine November & December 2023
Our next edition covers Prototyping and Production, so you'll have all the information you need to get that project idea from concept to schematic to PCB to product!

From the contents:
- PCB Services
- AI Tools for Developers
- BLE in Practice
- Low-Cost GNSS RTK Systems
- DIY Heating Plate
- ChatGPT Improves Firmware
- Open-Source Tools
- CNCing a Simple Enclosure
- Introduction to KiCad 7
- Solar Xmas Garland

And much more!

Elektor Magazine November / December 2023 edition will be published around **November 15th, 2023**. Arrival of printed copies with Elektor Gold Members is subject to transport. Contents and article titles subject to change.

Raspberry Pi Pico as Spectrum Analyzer

FFTs on a Low-Cost Hardware Basis

By Prof. Dr. Martin Ossmann (Germany)

This article shows how to build a software-based spectrum analyzer with a simple Raspberry Pi Pico. A basic version with a 12-bit resolution and a sampling rate of 500 kS/s is very well suited for audio measurements. With an external ADC, even 50 MS/s can be achieved!

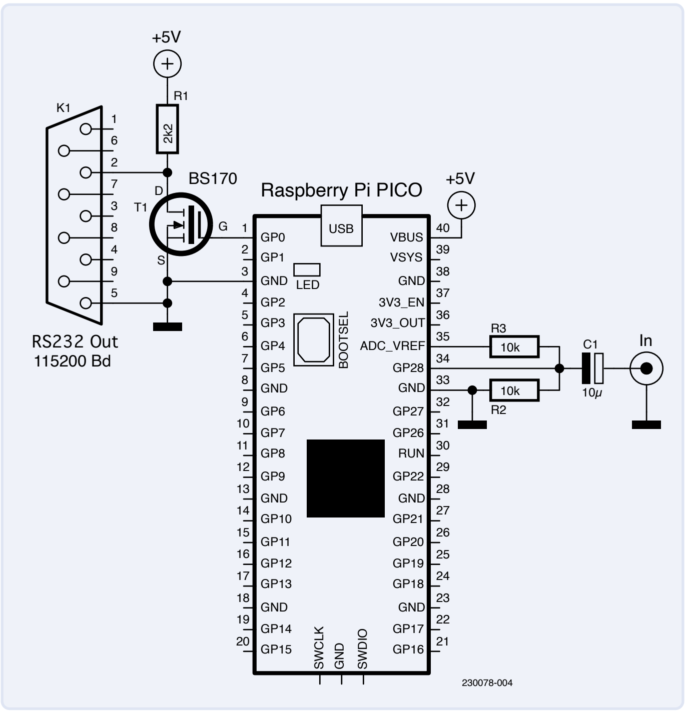


Figure 1: Minimal wiring of the Pico board for data transfer to the PC.

When using the Raspberry Pi RP2040 microcontroller's integrated ADC, analog signals are sampled with up to 500 kS/s and resolved at 12 bits. In the simplest case, a PC is used for the actual frequency analysis and display of the results. As the first expansion stage, an LCD can be connected, which already gives you a stand-alone analyzer. With an external ADC, up to 50 MS/s can be achieved in the next step, making the analyzer suitable for signals up to 25 MHz.

500 kS/s and PC Connection

For initial experiments, the Raspberry Pi Pico is wired as shown in **Figure 1**. R2 and R3 raise the ADC zero point to half the level range. C1 removes DC voltage components from the input signal. The input used, ADC2, uses Pin 34 together with GPIO28. T1 is used together with R1 to establish a TTL level for data output to the PC via the serial interface at 115,200 baud. Since the circuitry is quite simple, it can be easily implemented on a breadboard, as shown in **Figure 2**.

The ADC initially operates with the default sampling rate of 500 kHz. It transfers its 12-bit samples to a FIFO unit, from where they enter a buffer that can store *N* values. *N* typically has a value of 1,024. Whenever the buffer is full, the values are sent serially to the PC.

DFT and FFT

The spectrum is calculated using discrete Fourier transform (DFT). This algorithm takes *N* samples s_n (with $n = 0$ to N-1) of the input signal to calculate *N* spectrum values Z_k (with $k = 0$ to N-1). The transformation rule applied is the following:

$$Z_k = \sum_{n=0}^{N-1} s_n e^{-\left(\frac{2\pi ikn}{N}\right)} = \sum_{n=0}^{N-1} s_n \left[\cos\left(\frac{2\pi kn}{N}\right) - i\sin\left(\frac{2\pi kn}{N}\right)\right]$$

The DFT consists of N complex numbers. If the input signal is real, as in this case, the resulting DFT is symmetrical and the following applies:

$$Z_{N\text{-}k} = \overline{Z}_k$$

So only the first *N*/2 values are relevant. This is in accordance with the sampling theorem, according to which unique values are

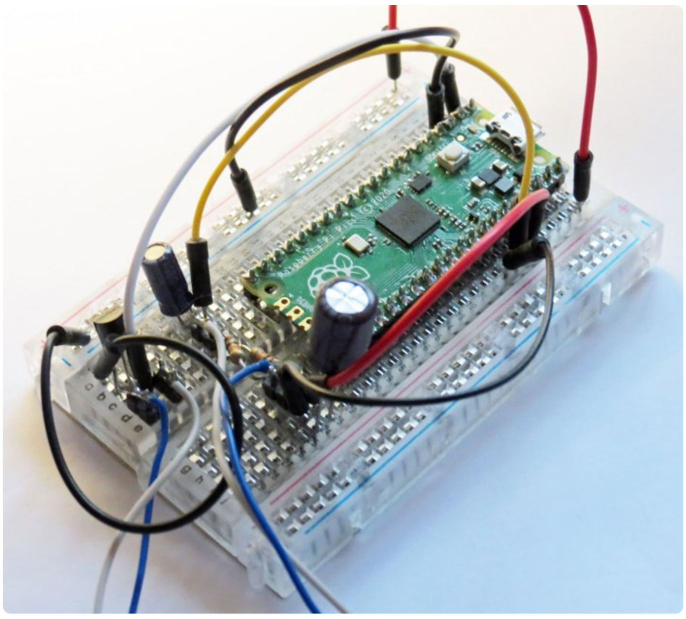


Figure 2: My breadboard design of the circuit from Figure 1.

obtained only up to half the sampling rate. Consequently, only the first $N/2$ spectral values are used in the following. The calculated spectrum is that of the input signal S continued with the period N. In order to understand why the DFT actually performs a spectral analysis, we first consider how the input signal $s_n = \cos(2\,\pi\, m\, n\,/\,N)$ is transformed with, for example, $m = 5$ and $N = 16$. This input signal is a cosine oscillation with the frequency $f = f_s\, m\,/\,N$.

The resulting transformed values are $Z_m = Z_5 = N/2 = 8$, and the other values, Z_k, are all zero. Thus, an oscillation of frequency m becomes noticeable exactly in the mth transformed value. If, instead of the cosine signal, a sine signal would be transformed, the imaginary part of Z_m would not be zero. But what happens when we transform a signal that is composed of oscillations of different frequencies? This is simple, too: The DFT is a linear transformation, which means that the DFT of a sum is also equal to the sum of the DFTs of the individual signals. The resulting formula is

$$DFT(u\,Un + v\,Vn) = u\,DFT(Un) + v\,DFT(Vn)$$

with the prefactors u and v.

If the input signal is composed of several oscillations $s_n = \cos(2\,\pi\, m\, n\,/\,N)$ with different frequencies m, this results in exactly the individual oscillations in the spectrum. A signal $s_n = A\cos(2\,\pi\, m\, n\,/\,N)$ with amplitude A results in a value $Z_m = A\,N\,/\,2$ in the spectrum. This value depends on the number of samples. To eliminate this, the DFT is scaled by a factor of $2\,/\,N$. The new spectrum then has the values $W_k = 2\,Z_k\,/\,N$. A cosine signal with amplitude A then results in a spectrum value of exactly A. The amplitude value A is the peak value of the cosine signal. For the effective value, it must be divided by $\sqrt{2}$.

Computing Time

The computing time of the DFT grows quadratically with the number N of samples. It can therefore take quite a long time with a larger N. Measurements with the Raspberry Pi Pico led to the approximation $T_{DFT} \approx 9\,N^2\,\mu s$. If N is a power of two, you can use the fast Fourier transform (FFT) instead of the DFT, since it calculates the same values much faster. On the Pico board, the computing time is $T_{FFT} \approx 4\,N\,\log_2(N)$. **Table 1** shows the computing times for some practicable values of N.

Table 1: Computing times for different N.

N	DFT in s	FFT in s
256	0.590	0.008
1,024	9.437	0.041
4,096	150.995	0.197
16,384	2,415.919	0.918

For a simple spectrum analysis, $N = 1024$ is often used. The calculation time of the DFT would still be acceptable here with approx. 9 s, but the FFT is much faster with 40 ms. Occasionally, $N = 16{,}384$ is used, too, for which the DFT would already take a good 40 minutes — unacceptably slow. The FFT, however, needs less than one second for this! N is therefore always selected as a power of two in favor of the FFT.

RMS Value and dBm

In communications engineering, quantities are often specified as effective values. Since for this the time-dependent quantity is squared, averaged and then the root is taken, the term root mean square (RMS) has become common for this. For a discrete, periodic signal with the period length N as with our DFT signals, the RMS value is calculated as follows:

$$RMS_s = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1}|s_n|^2}$$

It is also possible to calculate the RMS value of the spectral values. We then get:

$$RMS_Z = \sqrt{\frac{1}{N}\sum_{k=0}^{N-1}|Z_k|^2} \quad \text{and} \quad RMS_W = \sqrt{\frac{1}{N}\sum_{k=0}^{N-1}|W_k|^2}$$

Interestingly, the following now applies:

$$RMS_S = RMS_Z\,/\,\sqrt{N} \quad \text{or} \quad RMS_S = \sqrt{N}\cdot RMS_W\,/\,2$$

This means that the RMS value can also be calculated in the frequency domain instead of the time domain. For this, only the prefactor has to be considered. This relation is also called Parseval's identity [1]. The energy in the time domain is thus accurately reflected by the sum of the energies of the individual oscillations in the frequency domain. Energy or power ratios are often expressed in dB in communications engineering. For the power ratio $v = P_1\,/\,P_0$, we get $v_{dB} = 10\,\log_{10}(P_1\,/\,P_0)$. For absolute levels, the reference level

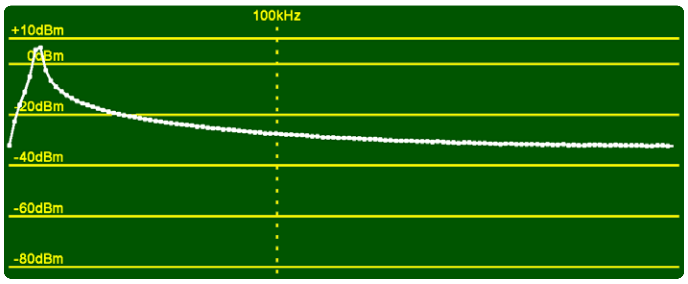


Figure 3: Poor spectral representation at half-integer frequency.

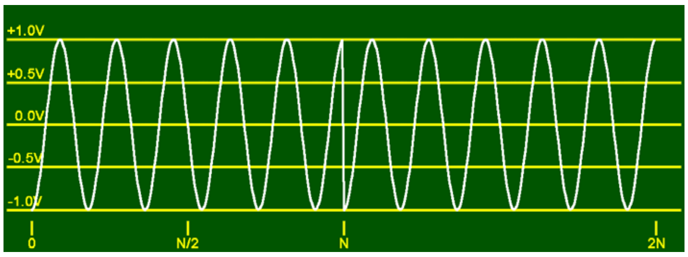


Figure 4: At sample N, an amplitude jump occurs if the signal is continued periodically.

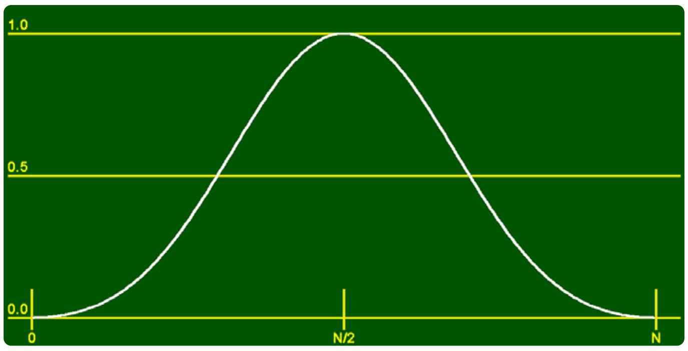


Figure 5: Blackman window function.

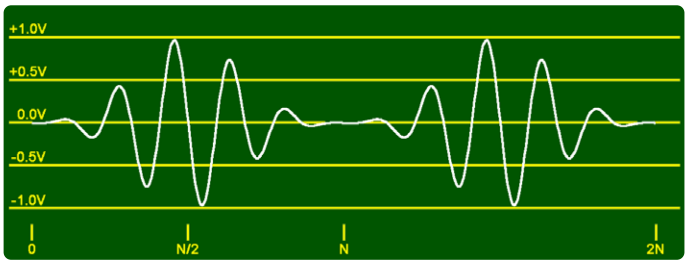


Figure 6: The signal multiplied by the window function has a continuous curve at the edges.

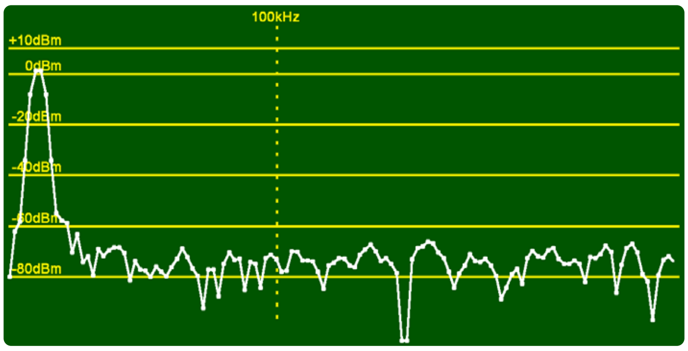


Figure 7: Spectrum after applying the window function.

$P_0 = 1\,mW$ is common. We then get $P_{dBm} = 10\,log_{10}(P\,/\,1\,mW)$. If we want to specify voltages U in dBm, we use the power P at a resistor of $R = 50\,\Omega$, at which the voltage U_{RMS} is applied. Since the power dissipated by the resistor is $P = U^2 / R$, we get $U_{dBm} = 10\,log_{10}(U^2_{RMS}\,/\,(R\,1\,mW)$.

For the spectrum analyzer described here, the Y axis is scaled in dBm, the usual logarithmic representation for spectrum analyzers, but note: In audio engineering, dBm often refers to a resistance of 600 Ω.

Signals and Windows

So far, we have considered input signals that fit exactly into a window of N values. The frequency m of the signal $s_n = A\,cos(2\,\pi\,m\,n\,/\,N)$ was therefore an integer. **Figure 3** shows the spectrum at $m = 5.5$ und $N = 256$.

Actually, one would expect a line-like spectrum because the signal contains only one frequency. Instead, you get a spectrum that falls off slowly on both sides of the maximum values. This is, of course, useless for a spectrum analyzer. The reason is that the periodically continued signal (period length N) contains a jump at the edges. This can be seen clearly in the amplitude/time diagram of two periods in **Figure 4**: At sample N, the signal jumps from +1 V to -1 V. This jump generates spectrally high-frequency components beyond the signal frequency. To avoid these effects, one applies the window technique, which in principle is equivalent to a superimposed envelope. This involves multiplying the input signal by a window function, which gently reduces it to amplitude 0 at the window boundaries, but otherwise does not change the signal much. **Figure 5** shows the window function used.

Figure 6 shows two periods of the signal multiplied by the window function. It can be seen clearly how the signal is amplitude-modulated so that it does not jump at the period boundaries (0, N, 2N...) but runs continuously. There are different window functions in use, each having different advantages and disadvantages. The following C listing shows how the Blackman window (three-term) used here is calculated:

```
double windowFun(int k) {
  double alpha=0.16;
  double a0=(1-alpha)/2;
  double a1=0.5;
  double a2=alpha/2;
  return a0-a1*cos(2*PI*k/(N-1))+a2*cos(4*PI*k/(N-1));
}
```

The window function is composed of two cosine functions. After applying the window, the FFT yields the spectrum of **Figure 7**. The spectral maximum is four bins wide around $m = 5.5$ and then drops significantly. With larger values for N, the spectral broadening is well acceptable and good, line-like spectra are obtained. However, we must consider that the window function reduces the signal amplitude. As window factor, we can approximate by using the arithmetic mean value of the window function values, as in the following listing.

```
float getWindowFactor() {
  float mean=0;
  for (k=0; k < N ; k++) {
    mean += abs(windowFun(k));
  }
 return mean/N ;
}
```

To compensate for the amplitude reduction, we divide the spectrum by the window factor after calculating the FFT. When using a window function, Parseval's identity no longer applies.

Analyzer Options

The spectrum analyzer offers several configurable options. In the most simple case, the µC samples the signal, calculates the spectrum, outputs it and repeats this endlessly. Sample rate f_s is equal to the sampling frequency. At the same time, it defines the frequency range of the spectrum because in principle this ranges from 0 to $f_s / 2$. Thus, at 500 kS/s, we can analyze frequency components up to 250 kHz. Signal length *N* determines how many samples are used for the spectrum, and thus defines the resolution of the FFT. Two adjacent spectral lines (called bins) have the distance $\Delta = f_s / N$. At 500 kS/s and $N = 1024$, the resolution is $\Delta = 500\ kHz / 1024 \approx 488\ Hz$. That's sufficient to resolve broadcast RF signals.

When analyzing signals with noise components, the individual spectra are often very ragged. This can be improved by averaging over successive spectra. This procedure is activated by the analyzer software's "MEAN" option, which, by the way, can be downloaded free of charge from the Elektor website for this article at [2]. The spectrum of the DCF77 time signal transmitter can be used to see how beneficial averaging is. Here, the sampling frequency is 250 kHz, and N is 16,384. **Figure 8** shows the resulting spectrum in the range of ±5 kHz around the center frequency of 77.5 kHz.

The thin white line shows a single spectrum that is clearly noisy. The averaging (thick green curve) clearly shows that DCF77 has a noise spectrum. This is because DCF77 transmits the data with pseudo-random noise at a bit rate of $77.5\ kHz / 120 \approx 645.8\ Hz$, in addition to amplitude modulation (very narrow-band). The bandwidth of ±645.8 Hz is represented by the two vertical lines.

Next, the spectrum of the EFR transmitter (129.1 kHz) is analyzed. EFR transmits RTTY data with a shift of ±170 Hz and a data rate of 200 Bd. Thus, two spectral lines should be seen in the spectrum at 128.93 kHz and 129.27 kHz. However, since only short messages are sent with long pauses between them, the frequency of +170 Hz occurs only rarely and is therefore difficult to detect. For this purpose, the software offers the *MAXHOLD* function, where the maximum of the spectra is calculated over many spectra. That way, the spectral components occurring briefly at 129.27 kHz show up more prominently (see **Figure 9**).

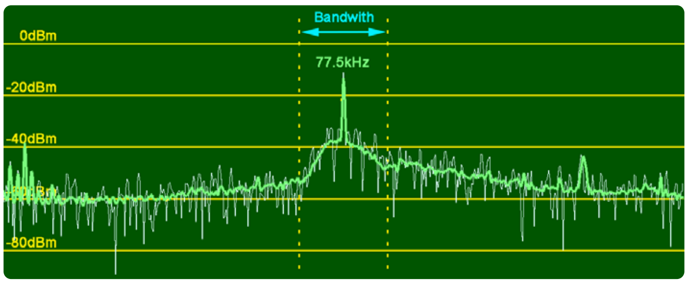


Figure 8: The "normal" DCF77 spectrum (white) and its averaged spectrum (green).

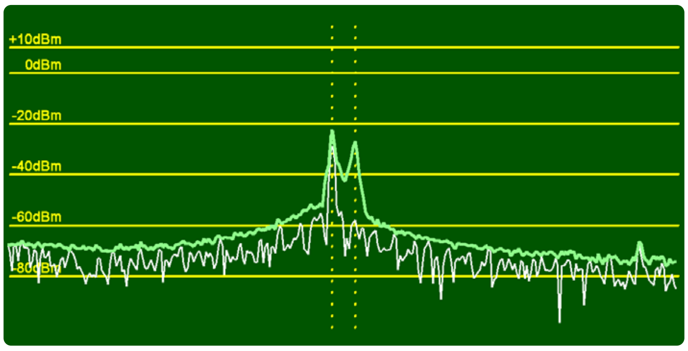


Figure 9: Spectrum of DCF49 (EFR at 129.1 kHz) with MAXHOLD function.

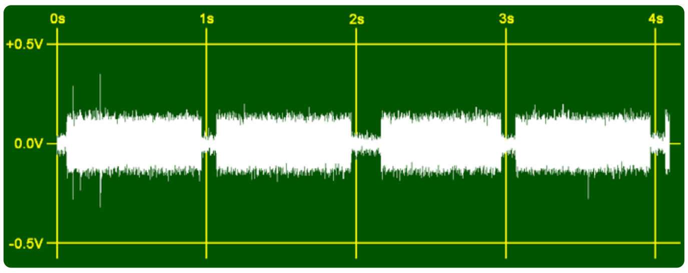


Figure 10: Amplitude over time of the DCF77 signal.

The thin white line is a single spectrum. Since no message is currently being sent, only the peak at 128.93 kHz is visible. The MAXHOLD spectrum is represented by the thicker green curve. The two peaks at 129.1 kHz ±170 Hz (at the dotted vertical lines) are clearly visible. To obtain such a finely resolved spectrum, many samples are required with $N = 16{,}384$. The sample rate is 500 kHz and the resolution, accordingly, is $\Delta = 500\ kHz / 16{,}384 \approx 30.5\ Hz$. Thus, the two spectral lines are just 340 Hz / 30.5 Hz ≈ 11 frequency samples away from each other.

Time Domain (Oscilloscope Function)

Of course, the signal in the time domain is also informative, if only to control how far the ADC acquisition range is utilized. The software can do this, too, by simply displaying the sampled signal. **Figure 10** shows a kind of oscillogram of the DCF77 signal at 77.5 kHz.

Since the carrier drops occurring every second are of particular interest here, it is advisable to select a lower sampling rate and a higher number of samples. For the representation in Figure 10, $f_s = 4\ kHz$ and $N = 16{,}384$ were chosen. The time window thus has a size of $T_s = N/f_s \approx 4\ s$, resulting in four carrier drops in Figure 10. The 77.5 kHz signal is heavily undersampled, but this is not a problem, since only the amplitude is relevant here.

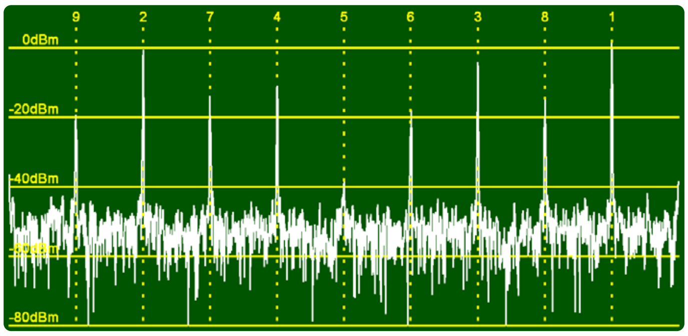


*Figure 11: Sub-sampling of a 225 kHz square wave signal with f_s = 500 kHz. A number of spectral lines show up at f_k = 225 kHz * k with k = 1, 2, 3...*

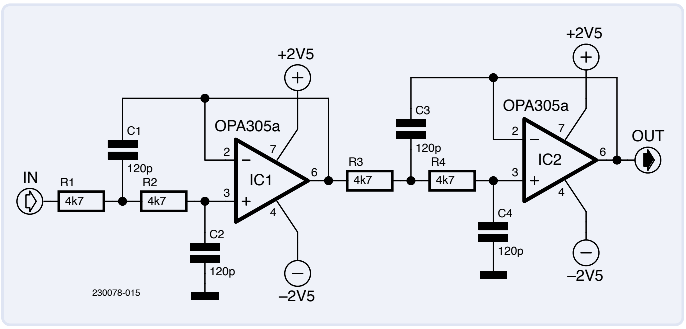


Figure 12: Anti-aliasing filter using two 2nd-order Sallen-Key filters with a cut-off frequency of 280 kHz.

Sampling Theorem

According to the sampling theorem, the signals to be analyzed may only contain frequencies up to a maximum of half the sampling rate. Either this is respected, or an appropriate low-pass filter is added before the ADC. Up to this point, no suitable filter has been suggested, since this always depends on the specific application. Furthermore, it is also interesting to study the conditions when analyzing higher frequencies. If we sample signals with infinitely short (Dirac) pulses, we get a periodic spectrum with period width = sampling rate. The spectral lines contained in the signal are thus periodically repeated with the width f_s. Accordingly, frequencies larger than f_s also appear in the spectrum. They are "folded" down into the fundamental interval, so to speak. If the sampling distance is longer, higher frequencies are attenuated more strongly. The relationships are illustrated in the following example: Sampling rate f_s is 500 kHz. A square wave signal with a frequency of 225 kHz and a duty cycle of 10% is analyzed.

In the spectrum in **Figure 11**, the fundamental wave at 225 kHz appears farthest to the right and has the highest level. The first harmonic ($k = 2$) has the frequency *2 * 225 kHz = 450 kHz*, which is already beyond $f_s / 2$. Due to the symmetry, it therefore appears at *f = 500 kHz - 450 kHz = 50 kHz* in the spectrum below the dotted vertical line labeled "2" with a still-considerable amplitude. The frequency for q is equal to 225 kHz * 9 = 2025 kHz. Since this is equivalent to 4 * 500 kHz + 25 kHz, this peak at 25 kHz appears on the far-left of the spectrum under the cursor line labeled "9." The cursor line indices *k = 1...9* represent the theoretically expected positions of the harmonics of the analyzed signal with frequencies of *k * 225 kHz*, known as "alias frequencies." Although the frequencies go up to over 2 MHz, ten peaks appear in the window extending up to 250 kHz. This means that the ADC can certainly handle higher frequencies than $f_s / 2$. This effect is used in what is known as undersampling. Before sampling, the signal can be filtered by a bandpass filter with a passband of $f_s / 2$ in such a way that only the relevant areas of the spectrum are captured. For this purpose, a fast sample-and-hold function would be added to the setup in order to still be able to process the high frequencies with the ADC. If this effect interfered, however, the signal would be filtered with a low-pass filter (antialiasing filter) to suppress higher-frequency signal components before sampling.

Antialiasing Filter

The effect of an antialiasing low-pass is demonstrated below. Here, a symmetrical square wave signal with 225 kHz and a duty cycle of 50% is filtered through the low-pass filter shown in **Figure 12** before the ADC. The filter was simulated with LTspice [3] and built with op-amps of the type OPA2350.

You can clearly see in the spectrum of the filtered signal shown in **Figure 13** that the harmonics are heavily attenuated and that only the 225 kHz fundamental wave stands out clearly above -40 dBm. However, our filter already attenuates quite heavily at 225 kHz. A filter with steeper edges would therefore be more ideal.

Noise and ENOB

So far, only signals composed of individual sinusoidal oscillations of different frequencies and amplitudes have been investigated. In practice, however, there are also noise signals with different bandwidths. Their treatment in terms of communications engineering is not exactly trivial, which is why only simple cases are described. As an example, noise values n_k with an amplitude of A are generated for binary noise with a noise frequency f_n. In each case, a die is rolled to determine whether $s_k = +A$ or $s_k = -A$ should apply for the next value. Initially, f_n = *1 MHz* applies, so that the values are generated faster than the ADC can sample them. Thus, the samples are independent of one another and the value is $\pm A$ in each case. Consequently, the noise signal is almost "white," which means that all frequencies occur equally in its spectrum. The signal is generated by software on the Raspberry Pi Pico. To enable this task to run in parallel with the spectrum analyzer software, it uses the second core of the RP2040 CPU. It is implemented using a PIO state machine, which allows up to 125 Mb/s to be generated.

As a rule, the RMS value of the noise voltage is commonly used. Since the signal is not periodic, we cannot formally use the formula for *RMS*. However, for large numbers of samples *M*, the RMS value can be approximated with:

$$RMS_n \approx \sqrt{\frac{1}{M}\sum_{k=0}^{M-1} n_k^2}$$

For binary noise, the RMS value can thus be calculated easily

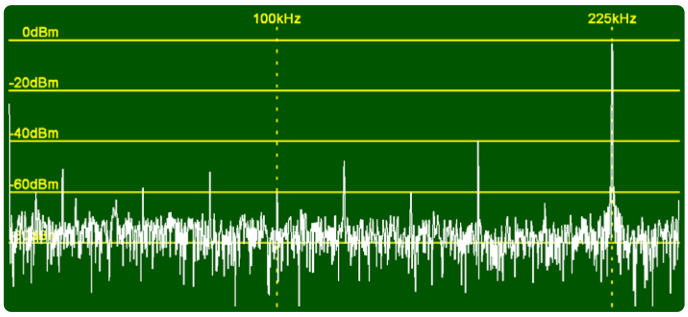


Figure 13: Spectrum of the square wave signal after antialiasing.

after all. Since each sample has the absolute value A, we get the simple equation $RMS_n = A$. In the test setup, $A = 750\ mV$ applies. **Figure 14** shows an example of this signal, and **Figure 15** shows the associated spectrum. The red thin line is a single spectrum. It is very jagged because it is only a single function specimen of the spectrum. The thick green line is the result of averaging over many spectra (MEAN function), and makes it more obvious that this is white noise at a certain level.

If we look at noise spectra with different signal lengths N, we can see that the displayed level also depends on N. The following relationship exists between the RMS value, the displayed level, and N: For a large N, the following applies: $RMS_s \approx RMS_n$. Due to Parseval's identity (see above), the following also applies:

$$RMS_W = 2\ RMS_S\ / \sqrt{N}$$

Now the energy in

$$RMS_W = \sqrt{\frac{1}{N}\sum_{k=0}^{N-1}|W_k|^2}$$

will be distributed evenly over w_k. For simplicity, it is assumed that all w_k are equal in size and equal to w. Putting all this together, we get: $w^2 = 4\ RMS^2_n / N$. Here, w is exactly the sought strength of the noise signal in the spectrum. In **Figure 15**, this level can be seen as a thick purple line. It is actually quite a good prediction of the real noise level. It also becomes clear why this level decreases with larger N: It is because the energy of the noise is distributed over more spectral lines.

In the next example, the bit rate of the noise source is reduced to $f_n = 25\ kHz$. Therefore, 20 successive samples are always equal, which is equivalent to a kind of low-pass filtering. The resultant spectrum is shown in **Figure 16**.

The thin red line again shows a single-specimen function. The averaged spectrum is shown in green. The first zero point of the spectrum is at 25 kHz. The spectrum is proportional to $H(f) = si(\pi\ f / f_N)$; where $si(x) = sin(x) / x$ is what is known as the gap function. Simplifying, one can calculate the level around 0 as follows: The bandwidth is approx. $f_{BW} = 12{,}5\ kHz$, which is equivalent to $k_{BW} = N\ f_{BW} / f_s = 25$ spectral lines.

If the spectrum has a constant bandwidth of w, the equation $RMS^2_w = 2\ k_{BW}\ w^2 / N$ is obtained for the RMS values by summing the spectral lines within the bandwidth. The factor of 2 is necessary because the FFT provides a symmetrical spectrum with the spectral lines $N - k_{BW}$ to N. Using Parseval's identity, spectral power w is calculated as follows: $w^2 = 2\ RMS^2_n / k_{BW}$. It is shown in Figure 16 by the thick purple line and matches the current spectrum quite well.

The next example is somewhat more practical: The ADC is now offered the signal of a sine generator with an amplitude of 3 V_{SS} (approx. 14 dBm). The signal frequency of $f_s = 48.828\ kHz$ corresponds exactly to the 100th spectral line. **Figure 17** shows the resultant spectrum.

The spectral line at f_s can be seen well and also has the correct size. At 2 f_s, we can see the first harmonic caused by the signal generator with a level of approx. 45 dB below the signal. Numerous small spectral lines can also be seen, since the components are not ideal and have small nonlinearities, for example. The noise level is also

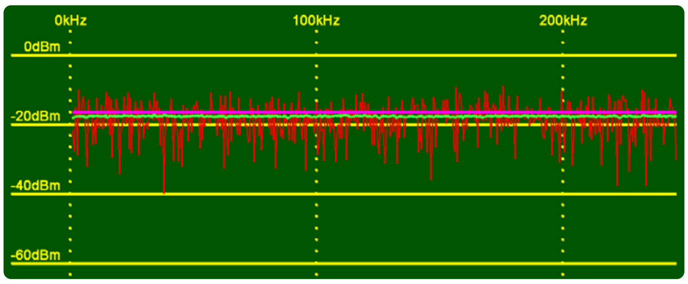


Figure 14: Binary noise as a time-domain signal with an amplitude of ±750 mV.

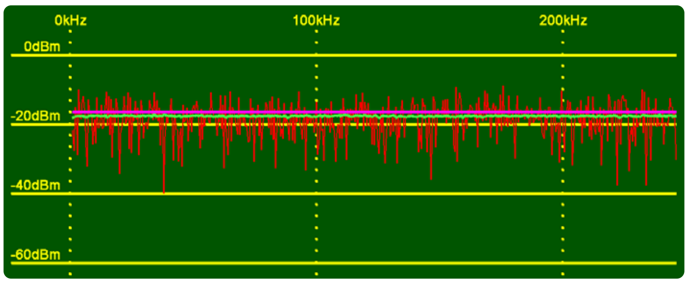


Figure 15: Spectrum of the binary noise in Figure 14.

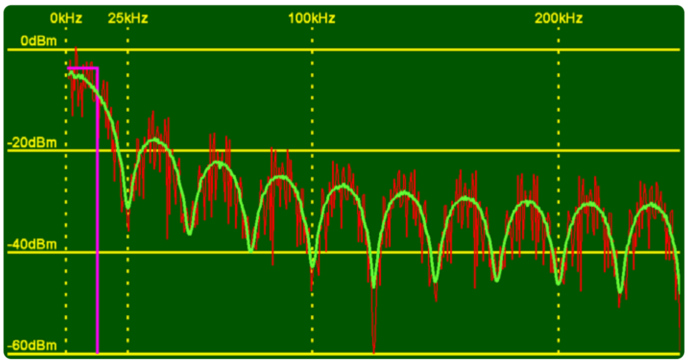


Figure 16: Spectrum of the noise at a bit rate of 25 kHz.

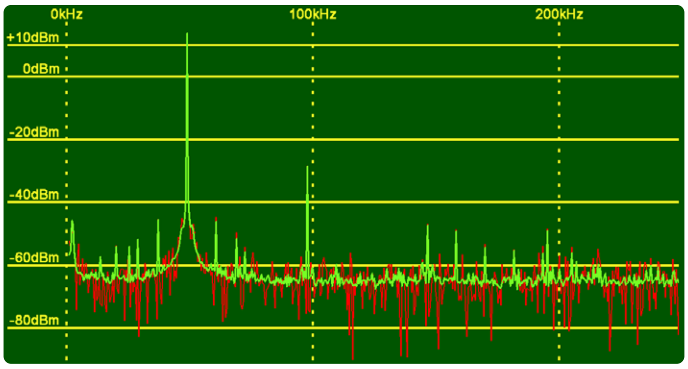


Figure 17: Spectrum of a sinusoidal signal with 48,828 kHz at a sampling rate of 500 kHz.

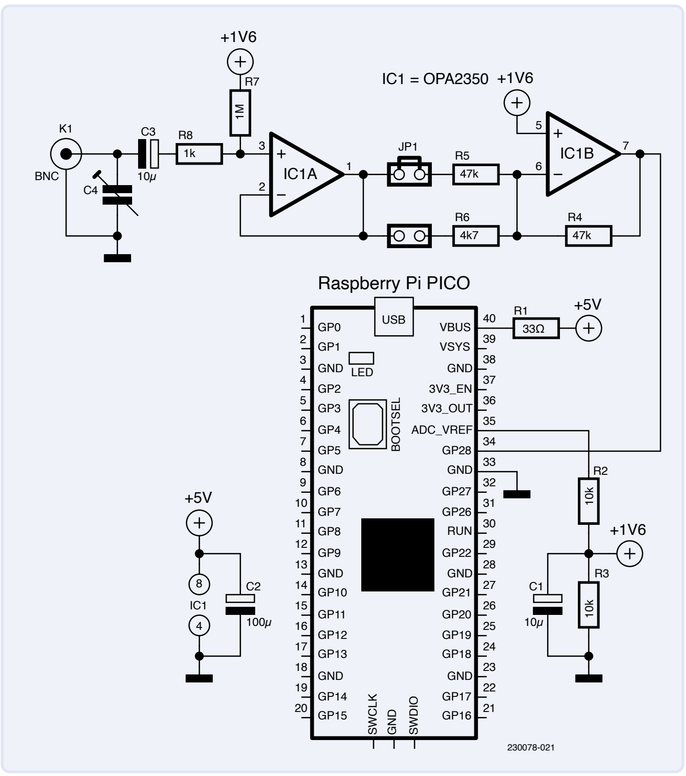


Figure 18: Preamplifier with an input resistance of 1 MΩ. For IC1, alternative op-amps AD823 or MCP602 may be used.

interesting. We can clearly see from the averaged spectrum (green) that this is about -65 dBm. With the formula given above, we can convert the level into the effective noise voltage of about 2 mV_{RMS} (approx. 5 mV_{SS}). The resolution of the ADC is *3.3 V / 4096 = 0.8 mV*. The noise thus covers about six quantization levels. More than 2 bits are thus lost due to the noise. Nevertheless, spectral peaks with -55 dBm still stand out well from the noise. The dynamic range is therefore at least *14 dBm - (-55 dBm) = 69 dB* - a factor of ≈2800. The actual resolution or the ENOB (**e**ffective **n**umber **o**f **b**its) of the ADC reaches more than 11 thanks to averaging, which is not bad at all. By the way, the noise experiments were done without windowing so that the levels are kept exactly and Parseval's identity is applicable.

Preamplifier with 1 MΩ Input

Oscilloscopes usually have an input resistance of 1 MΩ. This means that they normally do not put a significant load on the measurement objects and allow the usual 1:1 and 10:1 probes to be used. This convenience should also be offered by a spectrum analyzer, which is why the preamplifier shown in **Figure 18** was designed.

Opamp IC1A acts as an impedance converter. The input impedance is determined by R7. IC1B acts as a preamplifier with a gain of 1 or 10 selectable via JP1. IC1A must have a low bias current, otherwise this current at R7 would cause too large a voltage drop, thus moving the zero line of the signal away from the center of the ADC acquisition range. For example, the AD8042 type with its bias current $I_B = 1.2\,\mu A$ would result in an offset shift of $1.2\,\mu A * 1\,M\Omega = 1.2\,V$. Suitable opamp types are shown in Figure 18. With C4, an adaptation to a probe can be realized, if necessary. **Figure 19** shows the prototype built on a breadboard together with the Pico board and display.

ADC Clock and Sampling Rate

According to the data sheet, the ADC clock must be 48 MHz. This clock is normally generated by the USB PLL. The 16-bit prescaler

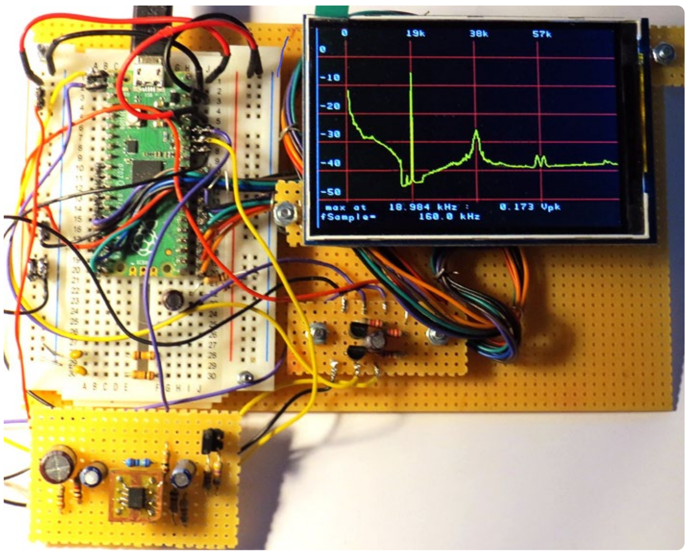


Figure 19: Analyzer consisting of a Pico board with LCD and a 1 MΩ preamplifier, showing a VHF multiplex signal.

is not activated, in this case (division by 1). The ADC is operated in *free-running mode*, in which it continuously acquires and outputs values. The timing is generated by an ADC counter which is triggered by the ADC clock. The ADC counter overflows at the value *ADCclkDiv* and then retriggers the ADC. Since a conversion requires 96 clock pulses, *ADCclkDiv ≥ 96* must be fulfilled. At the smallest value of *ADCclkDiv = 96*, we get the highest sample rate of $f_s = 48\,MHz / 96 = 500\,kHz$. The ADC counter is a fractional 16.8 counter: The integer part is 16 bits wide and you can fractionally divide by 8 bits to produce even uncommon sampling rates as accurately as possible. The slowest sampling rate is $f_s = 48\,MHz / 65{,}536 = 732\,Hz$.

In the next experiment, the ADC is operated far outside its specification for a sampling rate of 1 MHz. For this, the ADC clock must be 96 MHz. The USB PLL clock with its 48 MHz is not sufficient. The ADC is therefore clocked by the PLL for the system clock *SYS-PLL*. The clock rate generated for the processor is normally 125 MHz. However, if it is set to 96 MHz, it can be used to clock the ADC directly and enable a sample rate of 1 MHz. Unfortunately, the processor then runs about 23% slower. This is not a problem, however, because the CPU is not used to capacity by the software anyway.

A 400 kHz sinusoidal signal with an amplitude of 2 VSS is used as the test signal. **Figure 20** shows the resulting spectrum. The useful signal is shown with the correct level at the correct position. However, the first harmonic at 800 kHz is folded down to 1 MHz - 800 kHz = 200 kHz. The noise level of -65 dBm is not worse than at a sampling rate of 500 kHz. So, there seem to be no serious objections against clocking the ADC at 96 MHz.

LCD

Since fine details are often of interest in spectra, the MAR3502 Arduino Shield was chosen thanks to its relatively high resolution of 320×480 pixels. It is connected in parallel to the Pico board as shown in **Figure 21**. This allows a spectrum analyzer to be set up as a stand-alone device - but the controls are still missing.

Figure 20 shows the spectrum of an FM multiplex audio signal on the display. You can clearly see the mono signal (L+R) in the

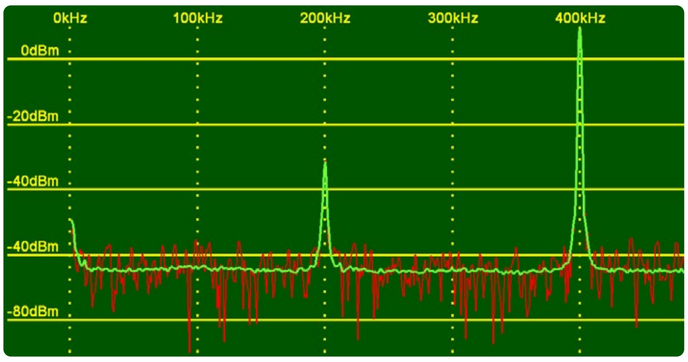


Figure 20: Spectrum of a 400 kHz sinusoidal signal at a sampling rate of 1 MHz.

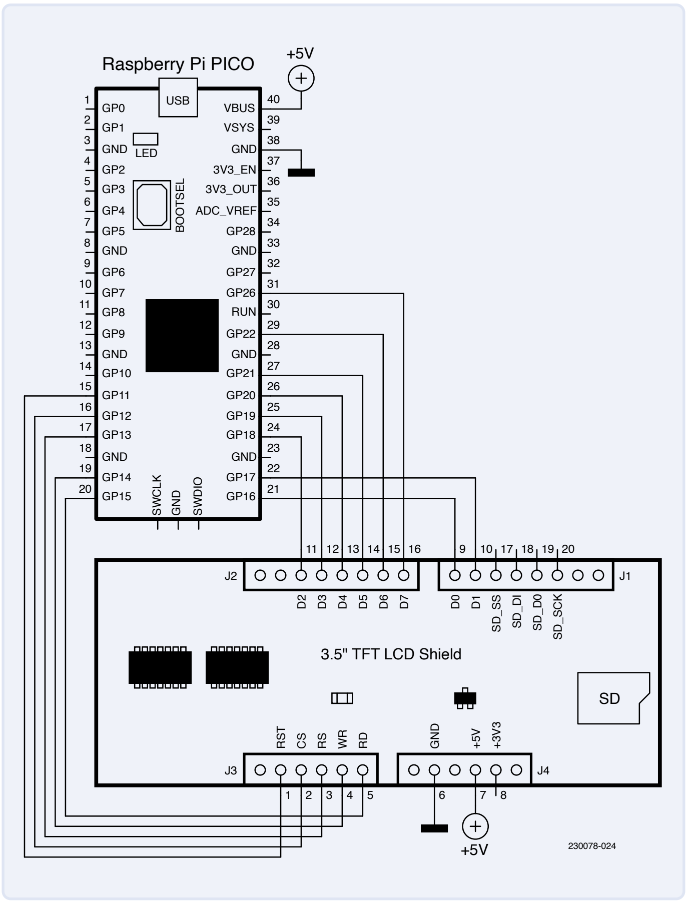


Figure 21: Parallel connection of a 3.5" 480×320-pixel LCD to the Pico board.

range at 17 kHz. At 19 kHz, the spectral line of the stereo pilot tone appears. At 38 kHz, the spectrum of the stereo difference signal (L - R) is visible. At 57 kHz, the spectrum of the RDS signal is visible, which consists of two sidebands around 57 kHz.

12-Bit, 50 MS/s ADC

For lower frequencies, the 500 kHz sampling rate of the ADC built into the RP2040 is sufficient, but there are also interesting spectra in the RF range. For this purpose, the Pico board can be extended with a fast, external ADC using the ADS807E chip. With this 12-bit ADC, up to 53 MS/s is possible. For the spectra, you need a lot of computing power and therefore some tricks to achieve this high sampling rate. Conveniently, the ADS807E provides a built-in reference voltage. The ADC is connected to the Pico Board as shown in **Figure 22**.

Due to the parallel connection, many GPIO pins are occupied, so you unfortunately can't connect the LCD at the same time. For this reason, the ADS807 gets a second Pico board. During the first experiments with it, the data is processed and displayed using the PC as in the beginning. Various analog front-ends can be connected to K2, which are presented below. **Figure 23** shows the test setup of a Pico board with an ADS807 and a 50 Ω transformer frontend.

By the way: If the ADS807 is too expensive, or you don't need its 12 bits, you can use an inexpensive 8-bit ADC such as the ADS830E, instead. It offers sample rates of up to 60 MS/s and only requires an adapted sampling routine.

Fast Sampling via PIO, FIFO, and DMA

The RP2040 CPU is normally clocked at 125 MHz. The following listing shows the minimal loop to acquire *N* data values with it:

```
for (k = 0; k < N; k++) {
  sampleBuffer[k] = gpio_get_all();
  gpio_put(ADCclock,0);
  gpio_put(ADCclock,1);
}
```

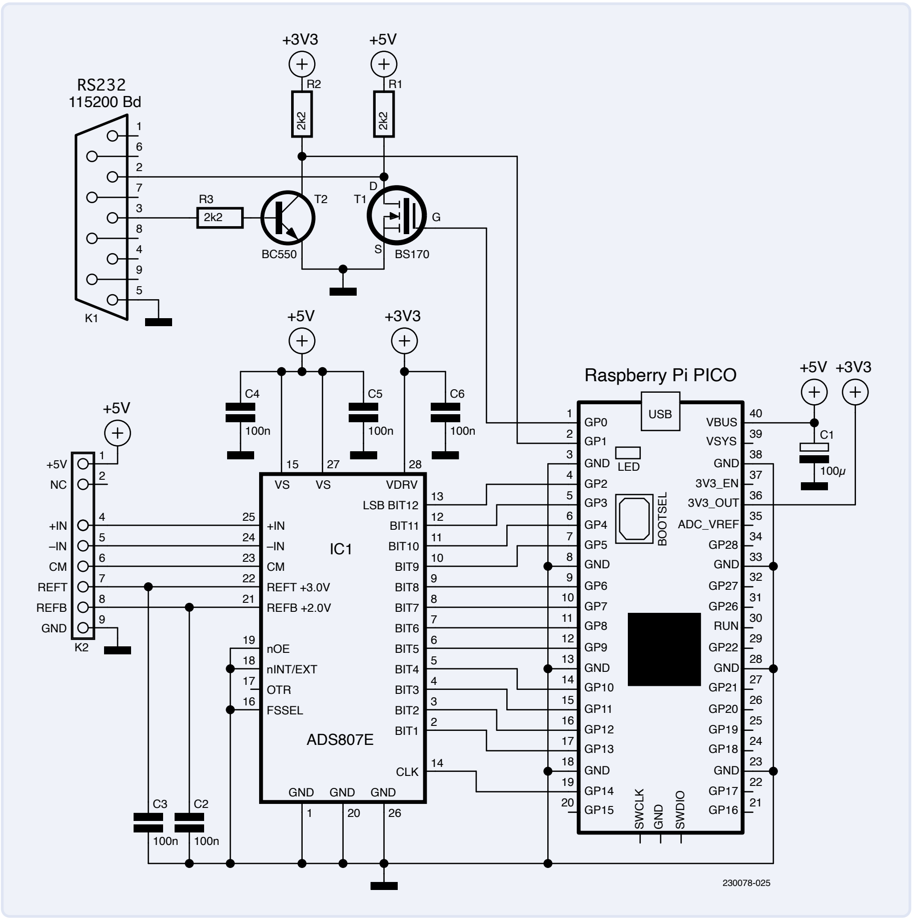


Figure 22: Connection of the external ADC ADS807 to the Pico board.

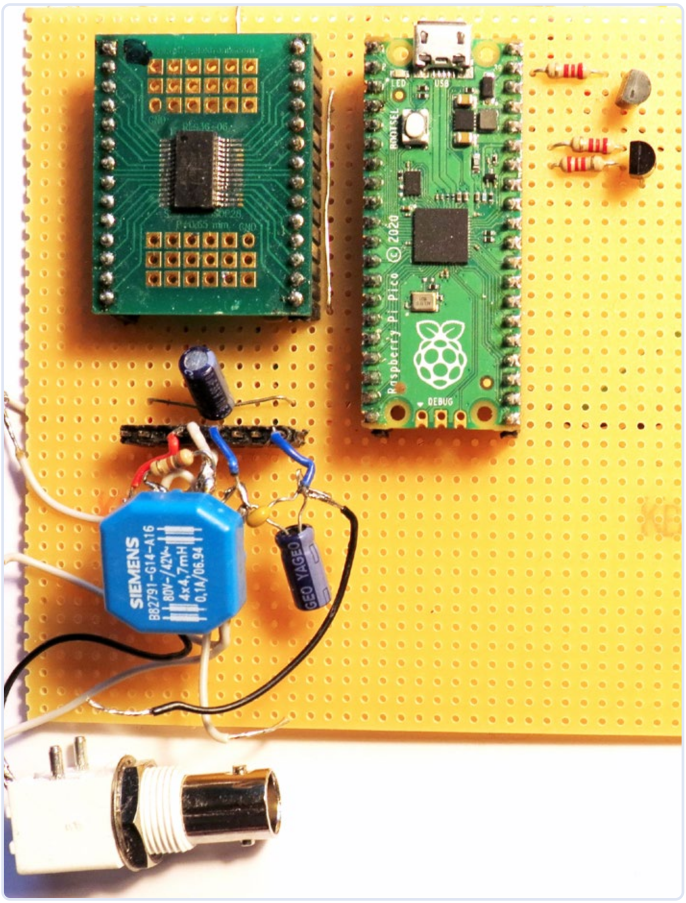


Figure 23: Extra Pico board with ADS807 and 50 Ω transformer frontend.

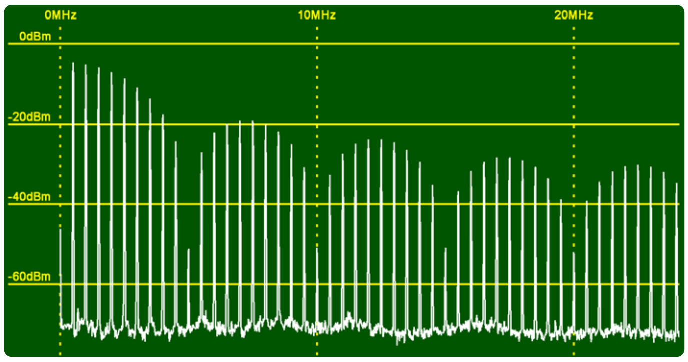


Figure 24: Spectrum of a 500 kHz square-wave signal with a duty cycle of 10%.

According to my measurement, the RP2040 CPU achieves a maximum of 15 MS/s - still quite a long way from the 53 MS/s of an ADS807. However, the CPU has a peripheral input-output (PIO) unit as a special feature, which consists of eight programmable state machines. These state automata can be controlled with the CPU clock and programmed with simple commands. In the following, some aspects of the sampling application are highlighted. The full details are quite complex, so please refer to the RP2040 documentation [4] for this.

The PIO program consists of only two commands:

```
.wrap_target
   in pins,12     side 0b0
   nop            side 0b1
.wrap
```

The keywords `.wrap_target` and `.wrap` cause the two instructions between them to be executed endlessly one after the other. The loop itself does not cost any overhead in the process. The `in` command causes 12 bits to be moved from the ADC into the ISR (**i**nput **s**hift **r**egister). As we know, the `nop` instruction does nothing. Following each instruction, the `side` option is used. With this, GPIO pins can be influenced in parallel to the actual PIO command. This way, the ADC clock is generated at GPIO14. Since each instruction requires one cycle, this program can, in principle, process 125 / 2 MS/s. This is sufficient for the ADS807. Besides the actual program, the configuration of the state machine plays an important role. It defines, for example, which pins are influenced by *side-set*. In addition, it was configured so that, after every two samples (when 24 bits have arrived in the ISR), the ISR value is written to an output FIFO. The data is quickly written to the memory via DMA (**d**irect **m**emory **a**ccess). The actual sampling therefore runs completely autonomously. The CPU only has to fetch the values from the buffer after each DMA transfer.

In principle, the same software used for the ADS807 ADC can be used for the integrated ADC. Only the sampling routine needs to be adapted. To achieve the maximum sampling rate of 53 MHz, for example, you can configure a CPU clock of 106 MHz and set the PIO prescaler to 1. Now, the PIO program runs at 106 MHz. Since each sample requires two clocks, the sampling rate is exactly 53 MHz. Unfortunately, the CPU is thus clocked a good 15% slower than usual. However, this is sufficient to achieve the tasks at hand without any problems.

Figure 24 shows the spectrum of a square wave signal with a duty cycle of 10% at a sampling rate of 50 MHz. The harmonics up to 25 MHz are clearly visible. Due to the duty cycle of 10%, every 10th harmonic is missing. Via subsampling, you can still analyze signal components up to approximately 200 MHz using the ADS807 without any problems.

Analog Front-Ends

Differential inputs +*IN* and -*IN* of the ADS807E have a high signal-to-noise ratio even at higher frequencies. However, the chip is somewhat more difficult to drive than ADCs with an unbalanced input. Possible frontends that can be connected to K2 in Figure 22 are described below. The dynamic range of the ADS807 is 2 V_{SS}, resulting in a resolution of 2 V / 4096 = 480 µV.

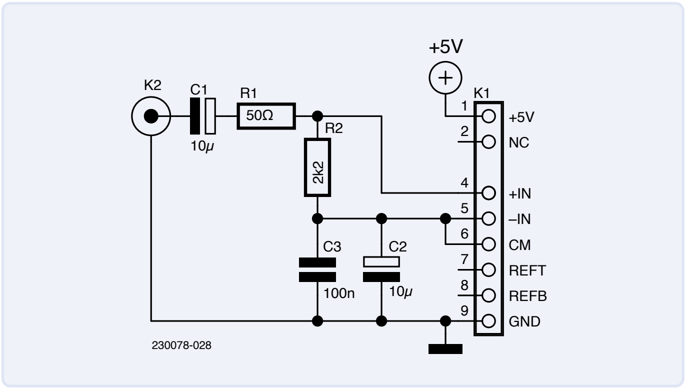


Figure 25: Single-ended frontend.

Front-end 1: single-ended

In the most simple case, you ignore the differential input, set *-IN* to CM (Common Mode) and simply feed the signal to *+IN*. You then work with AC coupling, and the DC offset of 2.5 V is supplied by CM (**Figure 25**). If you want to have a 50 Ω input impedance, you can simply connect a 50 Ω resistor in parallel to input K2.

Front-end 2: transformer coupling with 50 Ω

Using an RF transformer with three closely coupled windings with the same number of turns, a simple use of the differential input can be achieved. The prototype used a 4.7 mH common-mode choke with four windings, of which only three are used (**Figure 26**). The blue choke from Siemens can be seen clearly in Figure 23.

One winding each feeds the *+IN* and *-IN* inputs with opposite phases. The windings each have a parallel resistance of 100 Ω. The 1:1 coupling transforms these resistors into two parallel-connected 100 Ω resistors on the primary side, resulting in an input impedance of 50 Ω. Since the ADC sees twice the input voltage, this results in an amplification factor of 2. At the same time, the input is electrically isolated from the ADC, which avoids common-mode interference. The circuit is well suited for measurements on 50 Ω systems. If you want to wind transformer L1 yourself, you should wind the three windings in a trifilar way (i.e. one winding with three wires spooled together) to achieve a good coupling even at high frequencies. The core should have a high permeability even at frequencies up to 25 MHz.

Front-end 3: 25 MHz Preamplifier with 1 MΩ

For certain measurements, even a fast spectrum analyzer requires a high-impedance input. This then allows the usual 10:1 oscilloscope probes to be used. **Figure 27** shows the circuit of a suitable preamplifier.

Opamp IC1A serves as an impedance converter with an input resistance of 1 MΩ due to R5. The OPA2350 type is not only fast, but also has a low bias current and low input noise. IC2A and IC2B serve as bipolar drivers for the ADC. Since the unipolar input is converted into a differential signal, the result is an amplification factor of two. IC2 has been selected considering the high bandwidth so that the phase difference between *+IN* and *-IN* remains as small as possible. R2 and R4 are used for the stability of these opamps. IC1B buffers the offset voltage of 2.5 V. **Figure 28** shows, among other things, the setup of the frontend on a breadboard.

Noise

Certain noise sources make life difficult for a circuit designer. At R5, for example, a thermal noise voltage of $U_{N,\,RMS} = \sqrt{(4\,k_B\,T\,B\,R)}$ occurs, where k_B is the Boltzmann constant, T is the absolute temperature (in Kelvin), B is the bandwidth (here 25 MHz) and R is the value of R5 (1 MΩ). This results in an effective noise voltage of 6.3 mV.

This is several times the ADC resolution, and this noise level is strongly noticeable with broadband, high-impedance measurements. This also explains why a low-impedance system of e.g. 50 Ω

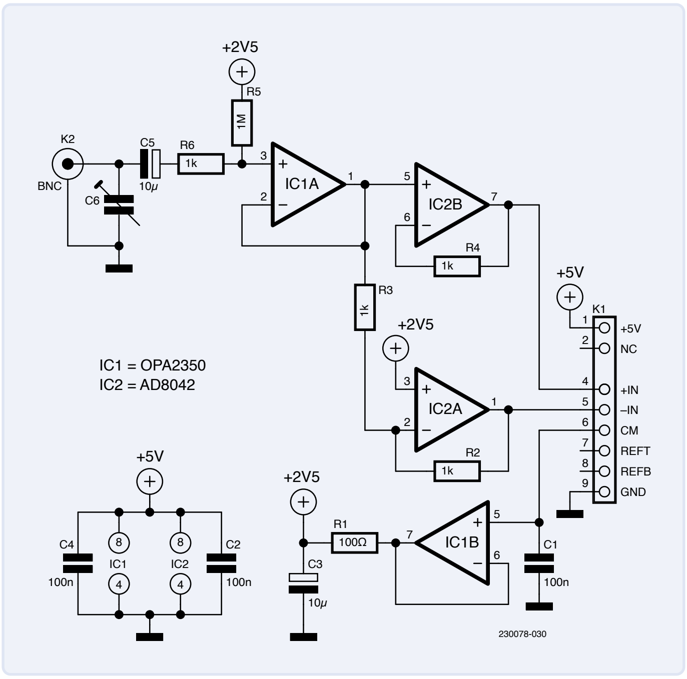


Figure 27: RF preamplifier with an input impedance of 1 MΩ.

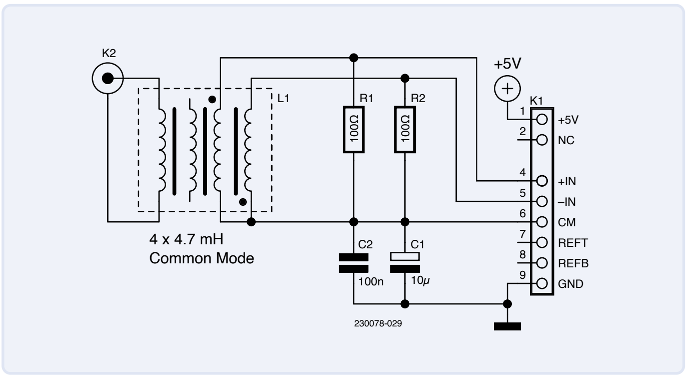


Figure 26: Transformer-coupled 50 Ω frontend.

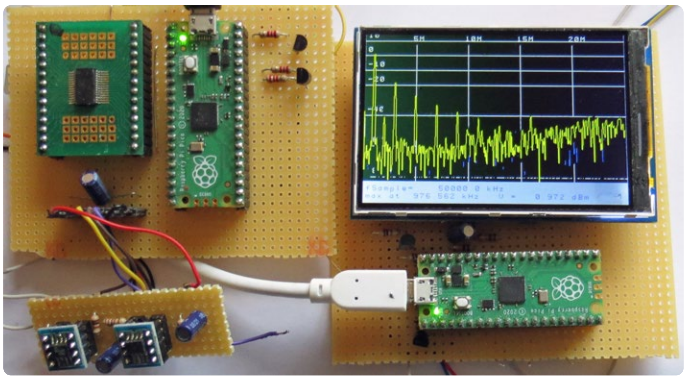


Figure 28: Stand-alone spectrum analyzer based on two Pico boards.

is more advantageous for RF. The input current of an OPA2350 has a noise density of 4 fA / √Hz. This results in a negligible effective noise voltage of 20 μV at R5 at a bandwidth of 25 MHz. For the input voltage noise, a density of 7 V / √Hz is specified. This results in a noise voltage of 30 μV at 25 MHz, which is also negligible. However, it becomes clear how the noise parameters of the opamp influence the design. For example, the AD8042 has a current noise of 500 fA / √Hz and is therefore unsuitable as an impedance converter at the input.

Graphic Display

As already mentioned, it is not possible to connect the external ADC and the LCD at the same time due to the lack of GPIO pins. For a standalone device, this shortcoming is compensated for by using two separate Pico boards: One for data acquisition from the external ADC, and one for the LCD. The latter serves as a graphics terminal that can independently perform various basic functions such as drawing lines and rectangles, as well as text output. On the Pico board with the ADS807, the spectrum analyzer functions are implemented. For graphics output, this part sends corresponding commands to the graphics terminal via the serial interface. At a price of a few dollars for a Pico board, this dual-board design is economically justifiable.

Figure 28 shows how this solution with two Pico boards might look. The display shows the spectrum of a 1 MHz square wave signal at a sample rate of 50 MS/s. The harmonics quickly become smaller with increasing order.

Translated from German by Jörg Starkmuth — 230078-01

Questions or Comments?

Do you have questions or comments about this article? Email the author at ossmann@fh-aachen.de or contact Elektor at editor@elektor.com.

About the Author

Martin Ossmann started reading Elektor at the age of 12 — and tinkering, of course. After studying electrical engineering and working for several years as a development engineer, he was a professor at the Department of Electrical Engineering and Information Technology at the FH Aachen University of Applied Sciences. He is not only the author of scientific publications, but has also been regularly publishing circuits and software projects with a lot of technical know-how in Elektor for more than three decades.

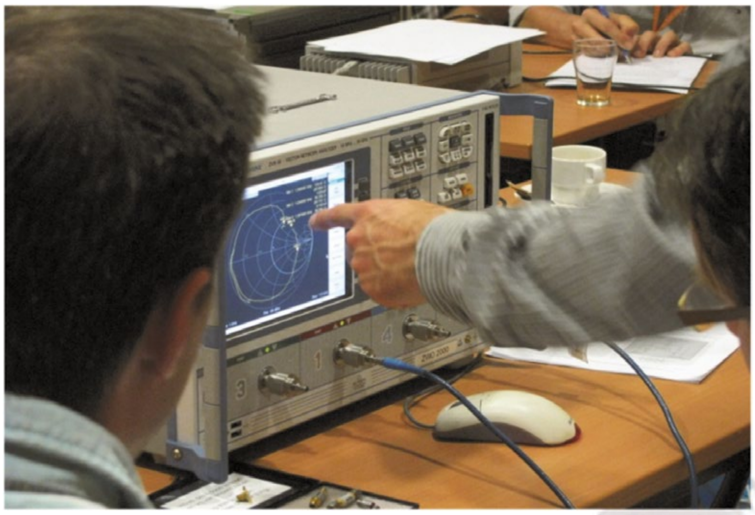
Related Products

- **Joy-IT JDS6600 Signal Generator & Frequency Counter**
  https://elektor.com/18714
- **OWON HDS242 2-channel Oscilloscope (40 MHz) + Multimeter**
  https://elektor.com/20415
- **Raspberry Pi Pico RP2040**
  https://elektor.com/19562

WEB LINKS

[1] Parseval's identity: https://en.wikipedia.org/wiki/Parseval%27s_identity
[2] Web page of this article: https://www.elektormagazine.com/230078-01
[3] LTspice: https://tinyurl.com/3zpwzk4y
[4] RP2040 data sheet: https://tinyurl.com/2sf4uvfm

Course Applied RF technology



Theory and practice of high-frequency systems
(in English language)

Date: 18 - 21 September 2023

Location: Dwingeloo, the Netherlands

More information on **www.astrotec.nl/Courses/RF_Eng** or via email: **rfcourse@astrotec.nl** or tel.: +31 (0)521 59 52 87.

Direct registration possible via **www.astrotec.nl/Courses/RF_Eng/registration**

Concept

The wireless equipment market has grown unimaginably. Smartphones, cable modems, anti-theft labels, remote data logging, embedded Bluetooth devices, wireless internet: radio frequency (RF) technology has permeated all aspects of everyday life.

Many technicians, including test and verification engineers, work with radio frequency systems. Understanding the coherence of system components is extremely important in order to be able to see the effects, causes and consequences of the influence various RF system components have on one another. Because high-frequency aspects play an increasingly important role in the design of embedded electronics, a course like this is also excellent as an introduction to digital / analog engineers who are or will be involved in the development of RF systems.

Good to know

The theoretical part of the course covers 75% of the course, with the remaining part spent in hands-on sessions. Part of the programme is a tour to the famous 14-dish Westerbork Synthesis Radio Telescope (WSRT).

The course will be held at ASTRON, which is located in the beautiful National Park Dwingelderveld, Europe's largest wet moorland area. ASTRON has price arrangements with several local accommodations, we can help you out finding the right place to stay.

This course is also taught in Dutch from 6-9 November 2023, check www.astrotec.nl/Courses/RF

Objectives

- better understand systems based on their high-frequency behavior
- recognize the importance of impedance matching in high-frequency systems
- describe the principles of modern communication systems
- understand specifications in data sheets
- describe various modulation techniques
- know the effects of non-linear systems
- understand the operation of various antennas
- be able to perform practical microwave measurements
- know the principles of RF measurement techniques and their limitations.

All RF courses incl.hands-on training are organized by ASTRON, Netherlands Institute for Radio Astronomy.

ASTRON

±40 V Linear Voltage Regulator

An Alternative Power Supply for the Fortissimo-100 Power Amplifier... and Others!

By Ton Giesberts (Elektor Labs)

For those who frown upon any shape or form of switched-mode power supply (SMPS) for the high-end Fortissimo-100 Power Amplifier, this project yields a 500+ VA, linear, symmetrical voltage regulator marked by low dropout voltage, high output current, and excellent stability — all obtained from discrete components and built from a kit!

Bearing in mind that nearly all high-performance audio power amplifiers benefit from a stabilized supply voltage, this linear power supply is specifically designed for a symmetrical output voltage of ±40 V and peak currents of 13 A (15 A peak achievable). As an example, the average current drawn by a Fortissmo-100 amp driving a 3 Ω load is roughly 4 A per regulator.

Design Considerations

The Elektor Fortissimo-100 high-end audio power amplifier [1] was proven to work best with a **regulated** ±40 V power supply, ruling out a "bare-bones" supply consisting of a transformer, a (bridge) rectifier, and a set of thick reservoir capacitors. A switched-mode power supply may not quite fit the bill either but this is more a matter of personal taste as the SMPS800RE does a good job. Still, there may be compelling reasons to favor a linear regulator built from through-hole components only, like the amplifier itself.

For the voltage regulator to work without dropouts (output voltage dips), the input voltage of the circuit must exceed the output voltage by at least 3 V higher, or even more in case of mains voltage fluctuations. Compared to most SMPSes (with a wide AC voltage input range), a linear regulator is less efficient and a large power transformer is called for with a higher power rating than without the linear regulator.

Today, most off-the-shelf power ("mains") transformers are marked by standardized secondary voltages. To create ±40 VDC directly, a transformer rated at 2× 30 V is the most likely choice. The resulting no-load DC voltage is usually around 42 VDC, largely depending on internal regulation of the transformer and voltage drop across the rectifier diodes. In practice, the no-load output voltage of a power transformer is always a few percent higher than loaded. The next higher standard secondary voltage is 35 V, which results in approximately 49 to 50 VDC or more at low output power — close to 52 V was measured in a Labs test setup.

With an 8 Ω load on the power amplifier, the regulator requires a small smoothing capacitance only. The advantage of the larger ripple voltage is a somewhat lower power loss in the supply regulator(s). But, at lower impedances, the ripple should not go beyond the dropout voltage (43 V at 10 A). In a lab test, a 2× 35 V, 300 VA toroid (ring core) transformer with 20,000 µF worth of smoothing capacitance appeared sturdy enough to feed the regulator. The maximum (near-clipping) sine wave power at 20 Hz and 0.1% THD+N into a 3 Ω load caused a mere 1.8 V_{peak} of dropout at the supply output. Mind you, the continuous output power then is 227 watts into the 3-ohm load and the 300 VA transformer is slightly overloaded. This, however, was not enough to trip the Fortissimo-100's protection.

Theory of Operation

The basis of any voltage regulator is measuring the output voltage, comparing it to a reference level, and controlling the output stage accordingly to counteract any changes. Although the present regulator circuit follows that concept, one dissimilarity that is marked is its much higher secondary reference voltage, which, at a little over 33 V, is relatively close to the 40 V target output voltage. The higher the reference voltage — 33.6 V here — the more gain left to a (simple) circuit to increase both the ripple rejection of the input voltage and the output voltage regulation.

Simply put, the circuit consists of a reference voltage, a differential amplifier, and an output buffer. Additionally, a safe operating area (SOA) protection is added to both regulators. Let's look at **Figure 1** to explore the operation of the **positive** regulator.

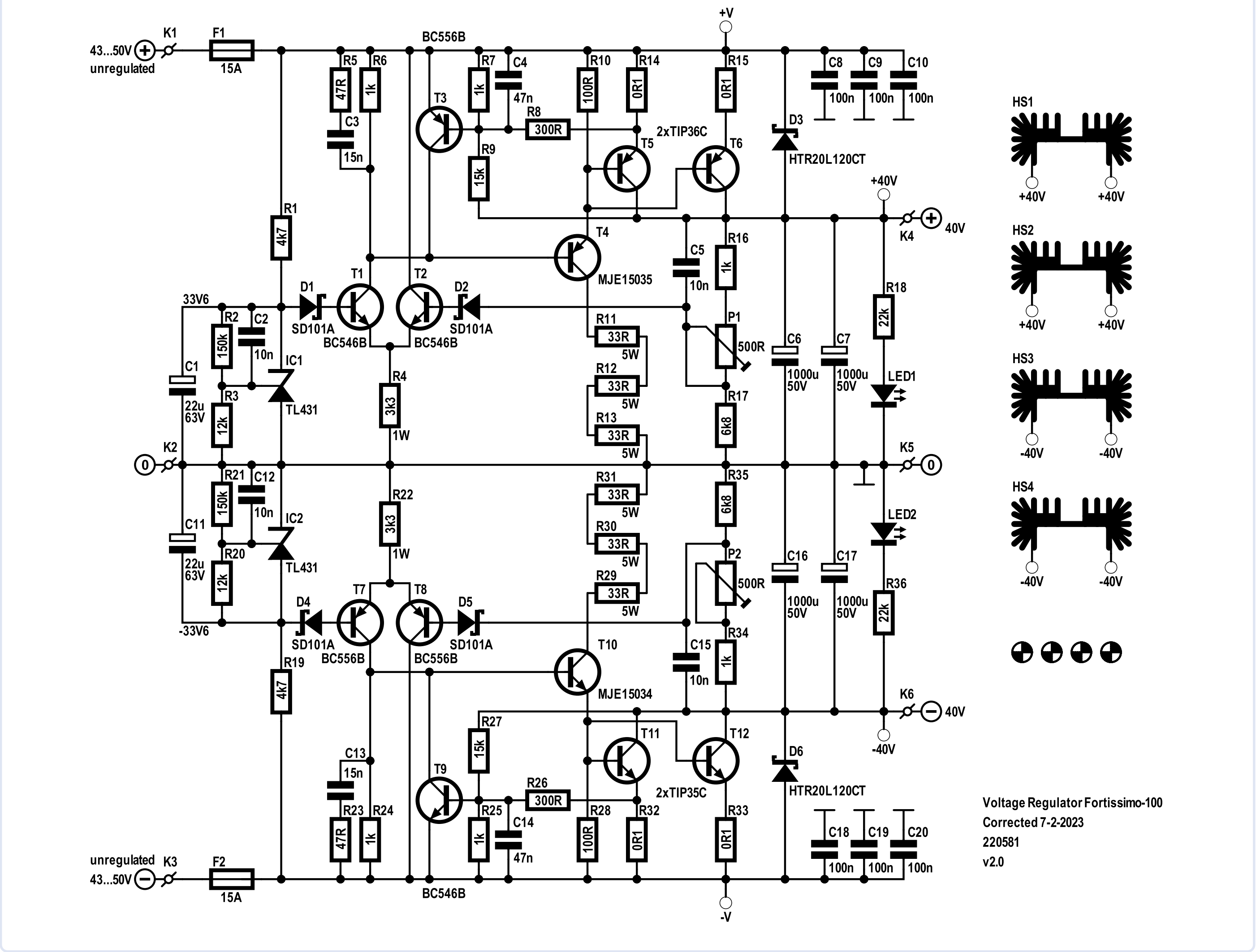


Figure 1: Schematic of the ±40 V linear voltage regulator, principally, but not exclusively, intended for the Elektor Fortissimo-100 audio power amplifier.

Quick Specifications

Input voltage range	52 VDC (low-power usage) to 43 VDC
Output voltage range	approx. 38.9 VDC to 41.4 VDC (theoretically) 38.6 VDC to 41.1 VDC (measured)
Dropout voltage at 6 A	42 V
Dropout voltage at 9.5 A	43 V
Dropout voltage at 13.5 A	44 V
Max. current	15 A peak (half sinewave) 4.8 A average
SOAR protection	15 A at 45 VDC in
Ripple rejection	>60 dB (@ 5 ADC load)
No-load input current	27 mA (@ 52 VDC input)
Assembly	from Elektor kit of parts; add power transformer, (bridge) rectifier, reservoir, capacitors

Reference Voltage

The reference voltage is not created by a Zener diode, as standard Zeners typically have considerable temperature coefficients. Special temperature-compensated versions are hard to come by these days, especially 33 V versions. Instead of a Zener diode, a type TL431 adjustable-precision shunt voltage reference is used with a maximum working voltage of 36 V. Its internal reference voltage (i.e., the primary reference voltage of the 40 V regulator) is typically 2.495 V. The cathode current through the TL431 is set by resistor R1. If the input voltage is between 43 and 50 V, the current is set between 1.9 and 3.4 mA, which turned out to be adequate for a stable 33.9 V reference voltage to be created. The 33.9 V is set by resistors R2 and R3, as follows:

$$V_{KA} = 2.495 \times (1 + R2 / R3) + I_{REF} \times R2$$

The TL431's adjust current, I_{REF}, is typically 1.8 µA, so the reference voltage is theoretically 33.95 V. However, this is specified at a cathode current of 10 mA, while in the prototype this current is lower and so is the voltage: 33.55 V was measured in practice. The TL431 is decoupled by C1 while C2 improves the overall stability.

Differential Amplifier

The differential amplifier is minimalistic, consisting of T1 and T2 with R4 as a current source. The voltage at the base of T1 is fairly constant. Ditto for the voltage across R3, even with the slightly varying voltage — with temperature — across the base-emitter junction of T1 and across D1. Schottky diodes D1 and D2 limit a (just conceivable) reverse voltage of T1 and T2's base-emitter voltages.

To reduce the influence of the voltage drop across each diode, while not affecting the input offset voltage of the differential pair too much due to changes in temperature, the pair is positioned next to each other on the PCB so both diode junctions are at the same temperature. A few millivolts — or even tens of millivolts — offset also caused by differences between T1 and T2 have no real effect on the much higher 40 V output voltage. Even an offset change of 30 mV means under 1% variance in the output voltage, which is inconsequential to the operation of the power amplifier.

The voltage across collector resistor R6 is used to drive the output stage. R5 and C3, as well as C4 and C5, form the frequency compensation to keep the regulator stable, even with SOA (safe operating area) protection T3/R7/R8/R9 active. Potential divider R16-P1-R17 measures the output voltage and presents the negative feedback for the differential amplifier. To compensate all tolerances, P1's output voltage range is approximately 38.6 V to 41.1 V. With the trim pot wiper at mid-travel, the output voltage will be pretty close to 40 V.

Output Stage

Although there are transistors that can handle the maximum output current called for when a constant 50 V input voltage is applied to the regulator, two transistors, T5/T6, are chosen to:

- limit the power dissipation per device to safe levels;
- increase the overload range;
- achieve lower dropout voltage and greater safe operating area.

Implementing these criteria reduces the risk of damage to the output stage in case there's an overload condition or even a short circuit. The larger PNP power transistors are type TIP36C (*cf.* the NPN TIP35C in the negative regulator) and are easily available from several manufacturers. PNP transistors are used in the positive regulator to keep the minimum voltage drop of the output stage as low as possible, the base currents flowing toward ground.

The dropout voltage is the sum of the transistors' saturation voltage and the voltage drop across the emitter resistors. A lower value for the emitter resistors would reduce the dropout voltage a little, but the currents through the two transistors can divert too much. At high collector currents, the transistors' gain is very low, and an additional transistor (T4) is needed to buffer the output of the differential stage. To prevent the saturation voltage of T4 increasing the dropout voltage of the output stage, its collector is connected to ground through a series connection of resistors.. This limits T4's power dissipation as well as its heatsink requirement. However, there's a catch in doing this: Should — by whatever cause — the input voltage fall below the dropout voltage, T4 will conduct permanently and the dissipation in its collector resistor will be quite high at 16 watts at 100 Ω total resistance and 40 V input voltage applied. This should never occur, though, hence three 5-watt resistors are used to prevent a burnout of this collector resistor.

An additional advantage of this collector resistor is its limiting of T5 and T6's base currents and thus acting as a simple current limit. The real protection, though, is formed by T3. The output current is measured by voltage divider R7/R8 as the voltage drop across the emitter resistor of T5, and this drives the base of T3. When, for instance, the current through R14 is about 7 A, the total output current is 14 A. The highest expected output current is a little over 12 A_{peak} with a 3 Ω load on the amplifier output. T3 will start conducting,

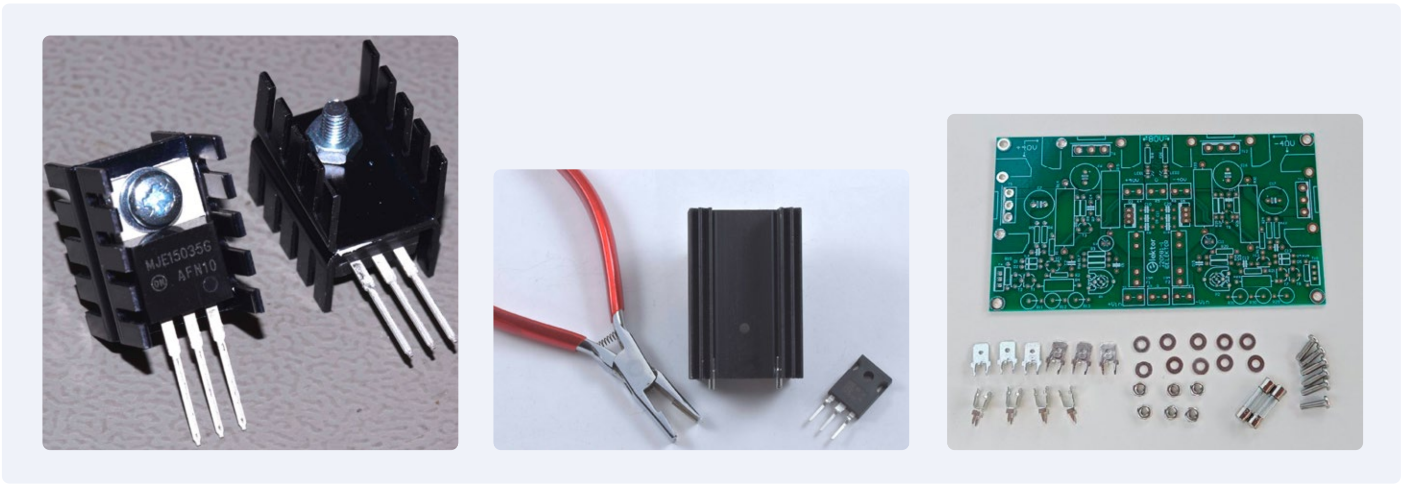


Figure 2: Some shots from the **Construction Manual** *written for the ±40 V linear voltage regulator project. The manual complements this article and is available for free download from [2].*

and — due to R9 — even sooner, depending on the voltage across T5. The exact level at which T3 is forward-biased is temperature-dependent and will be lower as temperature rises — an additional protection, and, with music, this won't be an issue.

D3 protects the output stage in case the input voltage is suddenly disconnected or short-circuited. T5 and T6 are decoupled with a pair of 1000 µF, low-ESR capacitors. LED1 indicates the presence of the +40 V output voltage. Although from the photos it appears that D6 is reverse-fitted on the PCB, in fact, both D6 and D3 can be fitted either way around and still function correctly. The HTR20L120CT diode in its 3-lead TO220 case has two internal diodes with a common cathode connected to the device's center lead.

The regulator input is protected by a 15 A fuse. The maximum RMS (root-mean-square) current must be considered, and at maximum half-wave sine current, the RMS value is $I_{peak}/2$, i.e., 6.5 A. However, at very low frequencies like 16.4 Hz (if you like organ music), the peak current may last several milliseconds. To make sure the fuse doesn't blow under such conditions, a 15 A type is used here, which, as a bonus, also reduces the voltage drop. If much more power is taken by the amplifier and/or the regulator, the primary fuse linked to the power transformer blows. The 15 A fuse will reliably blow should there be a sudden short circuit "behind" it.

Kit, Construction Manual, and Bill of Materials

The Elektor Store offers a comprehensive kit for the Linear Voltage Regulator project [2], containing the printed circuit board (PCB) and all parts listed in the bill of materials (BoM, or component list). This superb kit hopefully defeats readers' efforts in purchasing parts (electronic and mechanical) and having PCBs made to order.

Along with the kit comes a 12-page *Construction Manual* giving step-by-step instructions on assembling the project and hopefully reaching a perfect result. The manual is rich in drawings and photographs, a few of which are shown here as a collage in **Figure 2**. It also contains many tips and details on precise soldering, part positioning, tool handling, and simple mechanical work required to complete the project's build.

Since the proposed regulator isn't a complete power supply without the usual circuitry of a

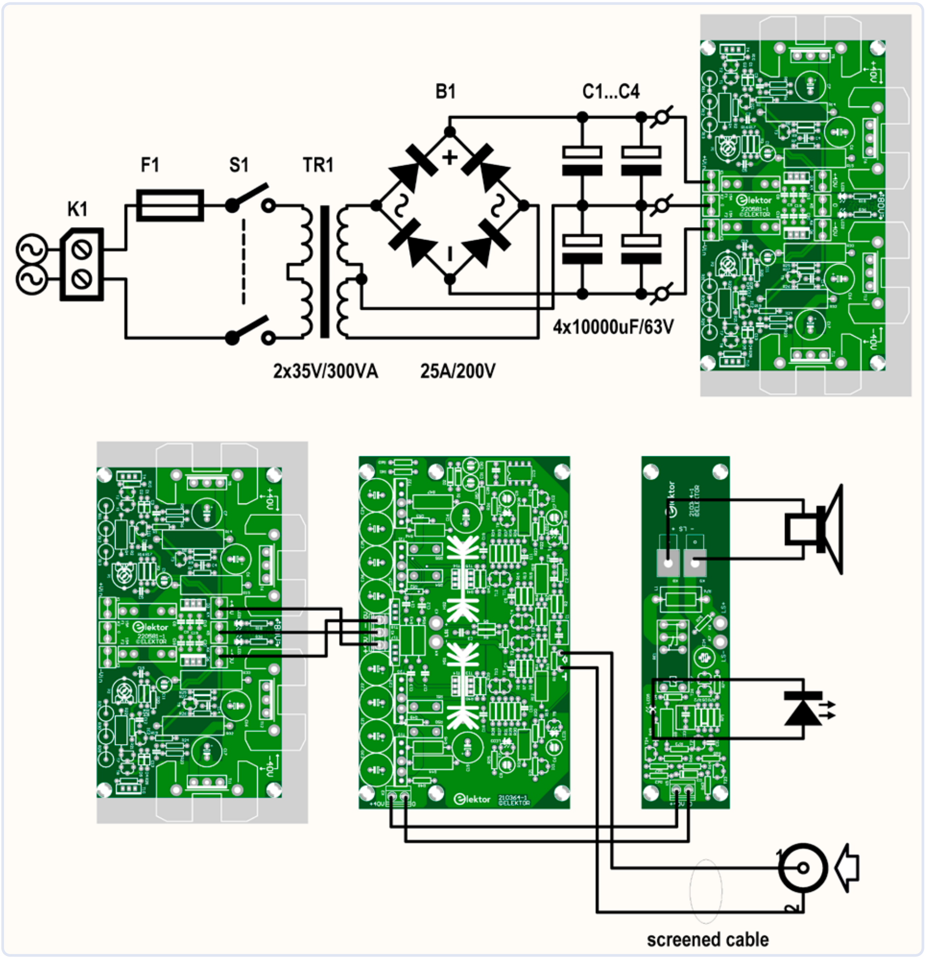
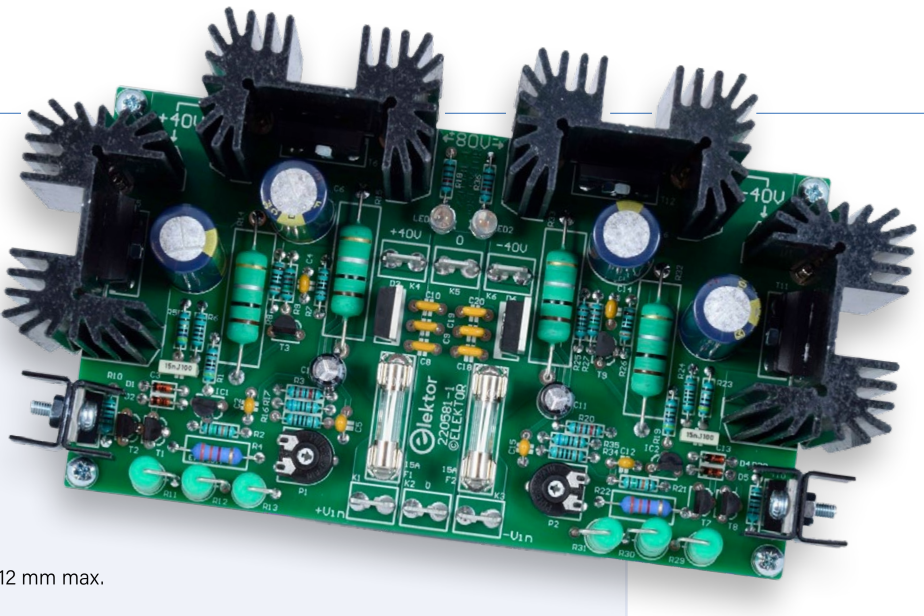


Figure 3: Suggested schematic for the unregulated power supply circuitry (top) and wiring diagram of the ±40 V linear voltage regulator / Fortissimo-100 combo (bottom).



Component List

(Elektor kit contents listed)

Resistors
R1, R19 = 4.7 kΩ, 1%, 0.6 W
R2, R21 = 150 kΩ, 1%, 0.6 W
R3, R20 = 12 kΩ, 1%, 0.6 W
R4, R22 = 3.3 kΩ, 5%, 1 W, body size 5×12 mm max.
R5, R23 = 47 Ω, 1%, 0.6 W
R6, R7, R16, R24, R25, R34 = 1 kΩ, 1%, 0.6 W
R8, R26 = 300 Ω, 1%, 0.6 W
R9, R27 = 15 kΩ, 1%, 0.6 W
R10, R28 = 100 Ω, 1%, 0.6 W
R11, R12, R13, R29, R30, R31 = 33 Ω, 5%, 5 W, body diam. 6.4 mm max. (axial, mounted upright)
R14, R15, R32, R33 = 0.1 Ω, 10%, 5 W (body 7.8×25 mm max.)
R17, R35 = 6.8 kΩ, 1%, 0.6 W
R18, R36 = 22 kΩ, 1%, 0.6 W
P1, P2 = 500 Ω, 0.15 W, trimmer, top adjust (Piher PT10LV10-501A2020-S)

Capacitors
C1, C11 = 22 µF, 20%, 63 V, pitch 2.5 mm, diam. 6.3 mm max.
C2, C5, C12, C15 = 10 nF, 10%, 100 V, ceramic X7R , pitch 5 mm
C3, C13 = 15 nF, 5%, 100 V, PET, pitch 5 mm
C4, C14 = 47 nF, 10%, 50 V, ceramic X7R, pitch 5 mm
C6, C7, C16, C17 = 1000 µF, 20%, 50 V, pitch 5 mm, D 12.5 mm, 5000 h @ 105 °C (EEUFC1H102L, Panasonic)
C8, C9, C10, C18, C19, C20 = 100 nF, 10%, 100 V, pitch 5 mm, ceramic X7R

Semiconductors
D1, D2, D4, D5 = SD101A SB00018/D8, DO-35
D3, D6 = HTR20L120CT, TO-220
LED1, LED2 = LED, green, 5 mm (T-1¾)
T1, T2, T9 = BC546B, TO-92
T3, T7, T8 = BC556B, TO-92
T4 = MJE15035, TO-220
T5, T6 = TIP36C, TO-247
T10 = MJE15034, TO-220
T11, T12 = TIP35C, TO-247
IC1, IC2 = TL431BCLPG, TO-92

Miscellaneous
K1, K2, K3, K4, K5, K6 = Faston PCB tab, two pins, hole diam. 1.4 mm, 6.35×0.83 mm
F1, F2 = Fuse clips, 20 A, Littelfuse 01000020Z, for 5×20 mm fuse
F1, F2 = Fuse, Cartridge, fast acting, 15 A , 5×20 mm
HS1, HS2, HS2, HS4 = heatsink MC33271 (for T5/T6/T11/T12), 2.7 °C/W
4× heatsink type FK231SA220, 24 K/W (for T4/T10, two each)
10× M3 washer, plain, steel
6× M3 screw, 10 mm, pan head
6× M3 nut

power transformer, a rectifier, and smoothing capacitors added, a suggested schematic — tuned to the Fortissimo-100 amp — is given in **Figure 3**. The parts for this section are not included in the ±40 V Linear Voltage Regulator kit and must be purchased locally.

Safety Consideration

Although the construction of the project and its practical use are detailed in the *Construction Manual*, we feel obliged to print the following safety notice in this article as well:

The large heatsinks are connected to the ±40 V output voltage, **not** to GND. Always remove the input voltage before touching or working on the regulator!

Results Achieved

At Elektor Labs, a test setup was built to verify the operation of the Fortissimo-100 amplifier in combination with the ±40 V Linear Voltage Regulator described here. Both units were built from their respective Elektor kits. The following key ingredients went into the unregulated supply section:

- 1 pc. TX-146-300-235 power transformer (300 VA, 2× 35 VAC secondary).
- 2 pcs. 10,000 µF electrolytic capacitor per supply voltage rail (i.e., 20 mF on each rail).
- 1 pc. SB352SBPC-style bridge rectifier, 35 A/200 V (25 A/100 V satisfactory).

At low output levels from the Fortissimo-100, the frequency spectrum shows that very small improvements can be achieved when compared with the SMPS800RE switched-mode power supply (**Figure 4**). The graph shows the frequency spectrum at 1 W into 8 Ω. The SMPS800RE's switching artifacts are gone, but the rest of the spectrum is essentially the same. The overall performance of the combo is impressive, with harmonic distortion plus noise as low as:

- 0.0007% (1 kHz, 1 W, 8 Ω, B = 22 kHz)
- 0.0013% (1 kHz, 1 W, 8 Ω, B = 80 kHz)

The ±40 V linear voltage regulator described here and available as a kit from Elektor is a good alternative to the best, yet affordable, switch-mode power supplies on the market today, and should satisfy those of you objecting, however lightly, to the concept or performance of "them @#!%^ switchers". Feel free to join the technical discussions on the ±40 V linear voltage regulator on the Elektor Labs page created for the project [3].

220581-01

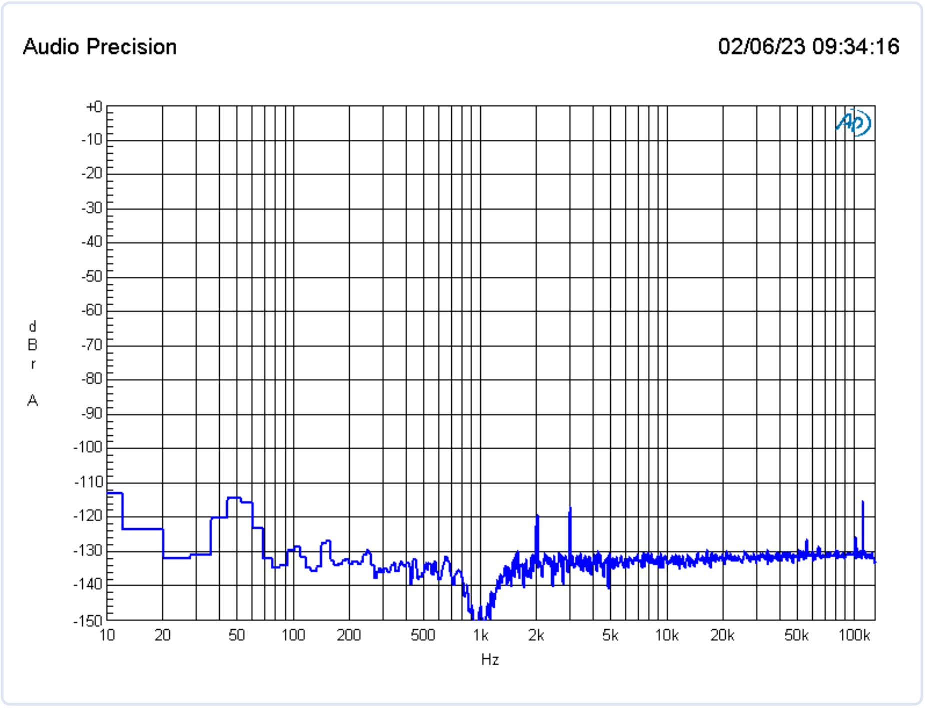


Figure 4: Here are the results! Fortissimo-100 output signal frequency while running 1 W into 8 Ω and powered by ±40 V Linear voltage regulator board no. 220581-1.

Questions or Comments?

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

Related Products

- **±40 V Linear Voltage Regulator Kit**
  https://elektor.com/20439

WEB LINKS

[1] Fortissimo-100 High-End Amplifier: https://elektormagazine.com/magazine/elektor-280/61057
[2] ±40 V Linear Voltage Regulator kit: https://elektor.com/elektor-40-v-linear-voltage-regulator-kit
[3] ±40 V Linear Voltage Regulator project on Elektor Labs website:
https://elektormagazine.com/labs/linear-voltage-regulator-for-fortissimo-100

MCU Wireless Communication Made Flexible

EEPROM Opens Networking Prospects for Wireless MCUs

By Gamal Labib (Egypt)

When connecting a microcontroller to a Wi-Fi network using the ESP8266, you might want a more flexible approach than hard-coded, fixed WLAN credentials. In this article, I will demonstrate a solution with a set of preferred AP networks to choose from interactively. Additionally, we can leverage the Wi-Fi Protected Setup (WPS) feature supported by many APs and routers.

The ESP8266 is a popular microcontroller unit (MCU) module that supports wireless communication and serves various applications in the Internet of Things (IoT). The boards based on this module usually have hard-coded Access Point (AP) credentials, such as Service Set Identifier (SSID) and passphrase, in their Arduino sketches. In some cases, developers also specify fixed IP settings. However, these hardcoded settings can cause issues when the wireless local area network (WLAN) topology changes. It becomes a headache return to the Arduino IDE or similar development environment just to update the sketches on deployed MCU boards, as the boards need to be recovered or replaced.

In this article, I will demonstrate a solution that provides flexibility in joining an MCU board to a WLAN. Instead of hard-coding a single set of WLAN credentials into project sketches, why not have a set of preferred AP networks to choose from interactively? By establishing an interactive dialog between the user and the MCU board through a touch screen, web page, or similar means, it becomes possible to specify new WLAN settings for the MCU board on the fly. Additionally, we can leverage the Wi-Fi Protected Setup (WPS) feature supported by many APs and routers. WPS allows an MCU board to join the user's preferred network at will. However, a question arises: Do we have to go through the joining steps of either the interactive dialog or the WPS every time the MCU board reboots? The answer is no, as long as we keep the newly specified WLAN settings in non-volatile memory accessible to the MCU. Fortunately, the ESP8266 has an internal Electrically Erasable Programmable Read-Only Memory (EEPROM) that can be manipulated by user code to store and retrieve data. I have utilized this feature to store up to 10 WLAN credentials set by either of the aforementioned methods. This approach not only provides networking flexibility but also enables the MCU board to connect to the WLAN with the best coverage out of the stored 10, if the developer chooses to do so.

However, it is important to note that extensive writing to the internal EEPROM is not recommended due to the expected lifespan of an EEPROM being based on the number of write cycles it undergoes. To work around this restriction and preserve the MCU's designated role, I have included an external EEPROM chip into the project hookup to make it possible for the user to explore doing the same job with an external EEPROM. However, for our WLAN configuration example here, we do not do extensive writing, so I opted for using the internal EEPROM.

Hardware

The Bill of Materials (BOM) for this project is quite compact. I used the WeMos D1 Mini, which is an ESP8266-based board known for its simple handling within the Arduino IDE and its small footprint. A 0.9" OLED graphic display module with a resolution of 128×64 pixels is used to display informative and debugging messages. It is connected to the MCU's I²C bus, along with an external 8 KB EEPROM chip. To select from the MCU's eight modes of operation (see **Table 1**), I used a combination of three DIP switches, which are wired to three of the WeMos's digital GPIOs (e.g., D5, D6, and D7). Since the DIP switches' connectors are tiny, they don't fit in the breadboard, so I constructed a daughter board to accommodate them. The project's breadboard implementation and screenshots of the WeMos in action are shown in **Figure 1**. The wiring diagram of the project components can be seen in **Figure 2**.

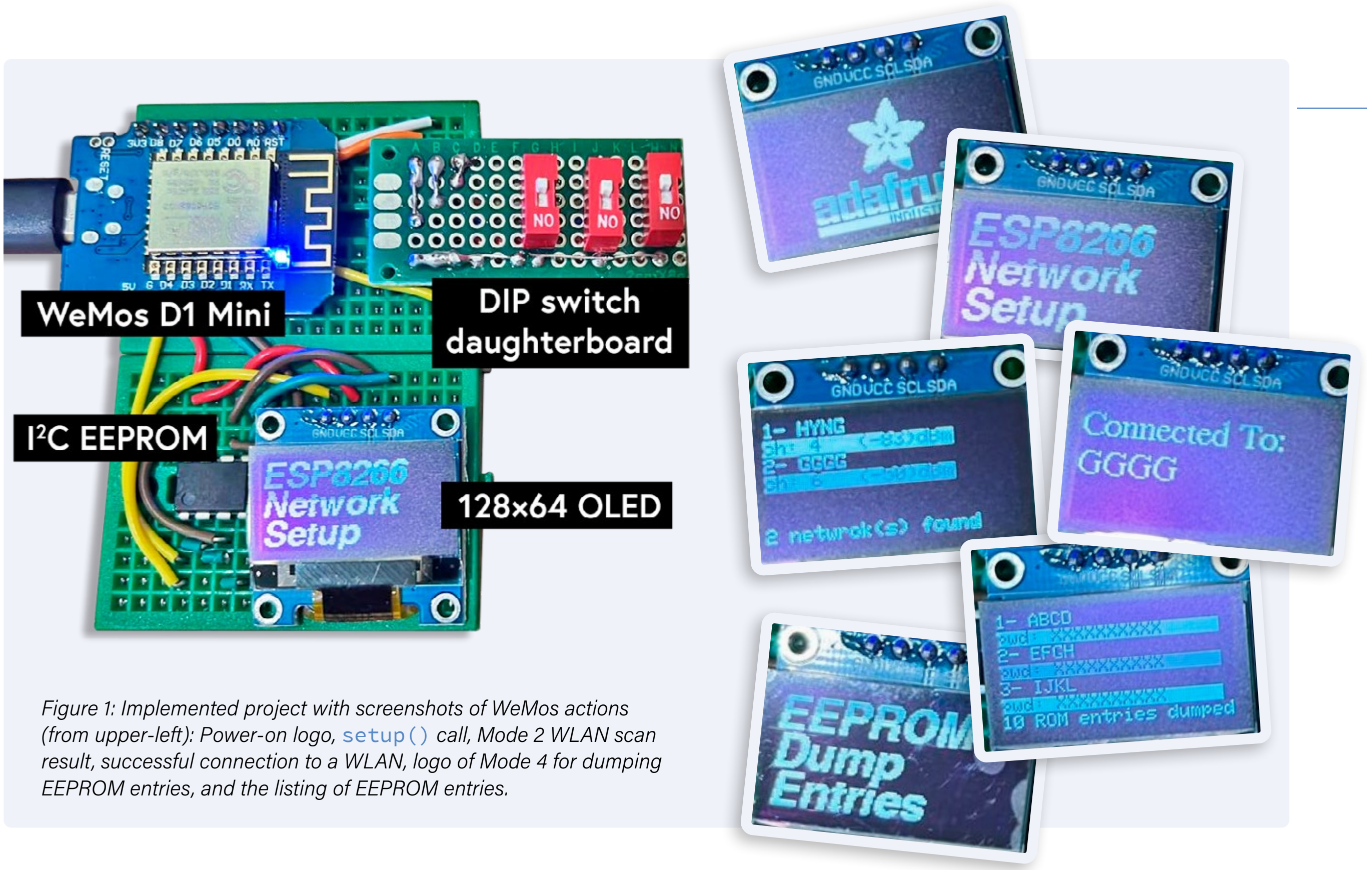


Figure 1: Implemented project with screenshots of WeMos actions (from upper-left): Power-on logo, `setup()` call, Mode 2 WLAN scan result, successful connection to a WLAN, logo of Mode 4 for dumping EEPROM entries, and the listing of EEPROM entries.

Table 1: DIP switch configuration of the relevant WeMos modes.

Mode	DIP 1	DIP 2	DIP 3	WeMos Action
1	0	0	0	Default WLAN selected
2	0	0	1	Scan available WLANs
3	0	1	0	Clear internal EEPROM
4	0	1	1	Dump internal EEPROM
5	1	0	0	Initialize internal EEPROM
6	1	0	1	Check external EEPROM
7	1	1	0	WPS
8	1	1	1	Interactive WLAN setup

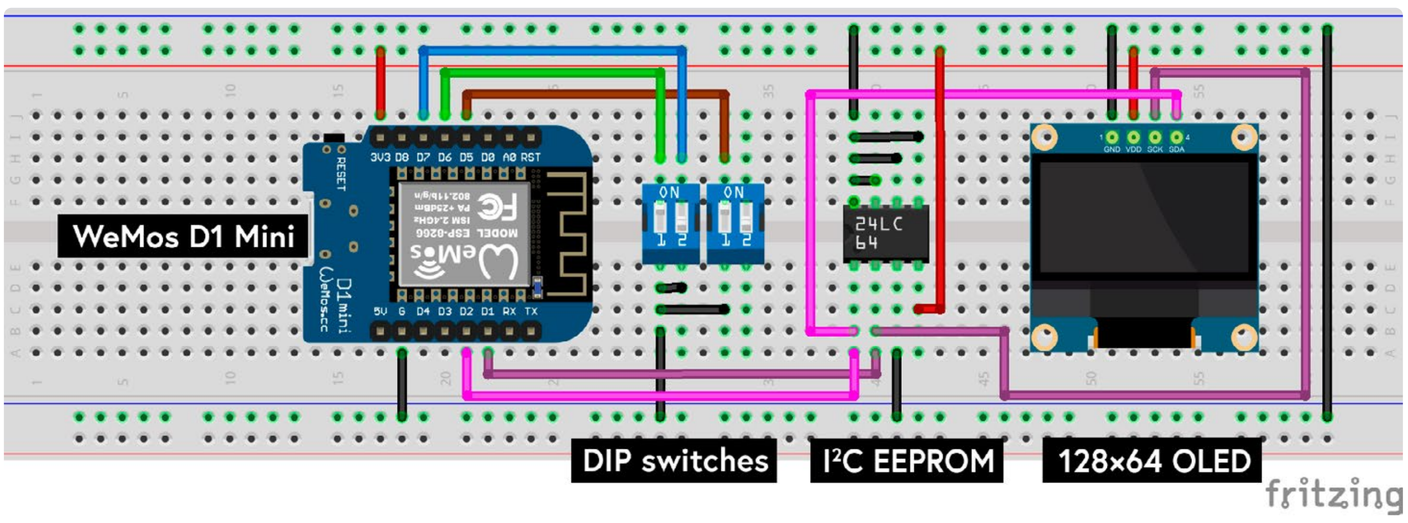


Figure 2: Project wiring using Fritzing breadboard tools and showing an extra fourth DIP switch for extended networking options.

Software

The project emphasizes software development best practices, such as well-structured and reusable code. To achieve this, I divided the 800-line sketch into six separate files, each handling a specific piece of hardware or functionality (see **Table 2**). This approach makes the code more manageable and helps readers understand the complex code involved. The main sketch file focuses on the WeMos networking modes within the `setup()` function, while leaving the `loop()` function free for the WeMos application code. During the development phase, I used `loop()` to manipulate the onboard LED blinking and ensure that the WeMos was functioning correctly.

Utilizing EEPROM for WLAN Networking

The ESP8266 module has 4 MB of flash memory, out of which 4 KB is set aside to emulate an EEPROM that is normally built-in on Arduino devices. Our example sketch stores the latest workable AP credentials in the internal EEPROM by default. With internal (or external) nonvolatile storage in use, our sketch can recall the AP credentials each time it boots after power-up. The EEPROM is also accessible by means of function calls from the Arduino's EEPROM library, and I'll follow this direction to save our network data that must be non-volatile for when the module is powered off. The main concept is to maintain the proposed structure shown in **Table 3** for holding WLAN AP credentials — as many as the internal or external EEPROM can hold. Each subsequent entry in the structure holds either the AP's SSID or its passphrase in 32 and 64 bytes of storage, respectively. That's 96 bytes in total per AP. Whenever the user adds a new WLAN to the list of preferred networks, the code appends this structure with the credentials of the new network's SSID and passphrase. The user needs to clear the EEPROM in the first run of the project code, using Mode 3, thus marking the end of the list with blank entries. The user will have the choice of applying the hard-coded WLAN credentials, using Mode 1, or checking the connectivity using the credentials in EEPROM, which is done using Mode 8. My implementation for this project is confined to this mechanism for up to 10 WLANs, limited by defined constant `MEMCNT` in the code, in addition to clearing the structure and initializing it with a preset list of AP credentials of the user's choice. The reader may increase or decrease the size of `MEMCNT` as necessary.

Background knowledge for accessing the EEPROM can be found at [2]. Now, I elaborate on the modes of operation for the WeMos, each of which requires the setting of the DIP switches (as per Table 1) then rebooting the WeMos to apply the setting.

Mode 1: Default WLAN Settings (Hard-Coded)

An MCU requires WLAN credentials, composed of an SSID and a passphrase for the relevant access point. Passing WLAN credentials can either be hard-coded in the flashed sketch or interactively passed to the MCU when the user is equipped with either a keyboard and a display or Serial Monitor in the IDE linked via the serial port. In this project, I'm using the conventional way of fixing AP credentials right in the sketch.

Table 2: Breakdown of Arduino code files and their functions.

.ino file	Description	Line Count
wemos-d1-mini-network	Global variables, setting up network mode	240
wlan-utils	Manipulate WLAN connections: WPS setup, scan/list WLANs, check connectivity	110
tft-utils	Manipulate TFT display	130
internal-eeprom-utils	Initialize EEPROM with five preset WLAN credentials, reset EEPROM, read/write entries	180
external-eeprom-utils	Check external EEPROM connectivity and size	40
ap-web-setup	Interactive WLAN setup: Initiate web server, display detected WLAN SSIDs, save selected WLAN to EEPROM	100

Table 3: EEPROM housekeeping table for multiple AP settings.

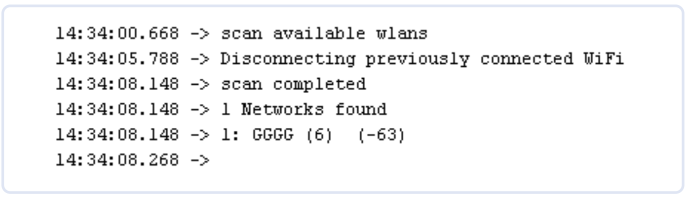
Entry Size (Bytes)	Description	Type
2	AP entry count	integer
32	AP1 SSID	string
64	AP1 Passphrase	string
.	.	.
.	.	.
.	.	.
32	APn SSID	string
64	APn passphrase	string

Mode 2: WLAN Scanning

In this mode, the WeMos disconnects from any connected WLAN in order to be able to scan for other accessible networks. Detected WLANs are listed by their SSIDs and signal strengths in decibels (dB). This mode gives the reader an insight into the prospects of networking the WeMos board and which mode to choose for joining a WLAN. **Figure 3** shows the outcome of booting the WeMos in this mode, where only one WLAN was detected and later on was joined, as shown in Figure 1d.

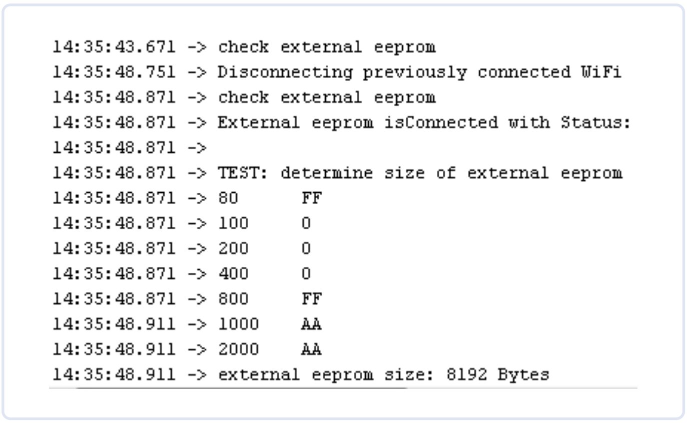
Mode 3: Clear Internal EEPROM

This mode causes the WeMos to erase its internal EEPROM in preparation for building a new list of preferred WLANs. All bytes in the EEPROM are cleared to 0×00 in this mode.



```
14:34:00.668 -> scan available wlans
14:34:05.788 -> Disconnecting previously connected WiFi
14:34:08.148 -> scan completed
14:34:08.148 -> 1 Networks found
14:34:08.148 -> 1: GGGG (6)  (-63)
14:34:08.268 ->
```

Figure 3: Arduino IDE's Serial Monitor messages — Mode 2 wireless network scan.



```
14:35:43.671 -> check external eeprom
14:35:48.751 -> Disconnecting previously connected WiFi
14:35:48.871 -> check external eeprom
14:35:48.871 -> External eeprom isConnected with Status:
14:35:48.871 ->
14:35:48.871 -> TEST: determine size of external eeprom
14:35:48.871 -> 80      FF
14:35:48.871 -> 100     0
14:35:48.871 -> 200     0
14:35:48.871 -> 400     0
14:35:48.871 -> 800     FF
14:35:48.911 -> 1000    AA
14:35:48.911 -> 2000    AA
14:35:48.911 -> external eeprom size: 8192 Bytes
```

Figure 4: Serial Monitor messages — Mode 6 checking external EEPROM sanity.

Mode 4: Dump Internal EEPROM

This mode causes the WeMos to dump the WLAN credentials structure, as explained in **Table 3**, from the internal EEPROM. Figure 1e and f show screenshots of the displayed EEPROM entries currently stored. A series of formatted SSIDs and passphrases is printed for the stored APs' credentials in this mode.

Mode 5: Initialize Internal EEPROM

This mode enables the reader to preset the credentials of 10 preferred WLANs into the internal EEPROM. This action is an exaggeration of the traditional hard-coded single WLAN credentials. Upon rebooting in Mode 8, the WeMos will inspect the list of credentials for WLAN connections. If its attempts fail to connect to any of preferred WLANs, then the reader would have to resort to either interactive or WPS modes.

Mode 6: Checking External EEPROM Status

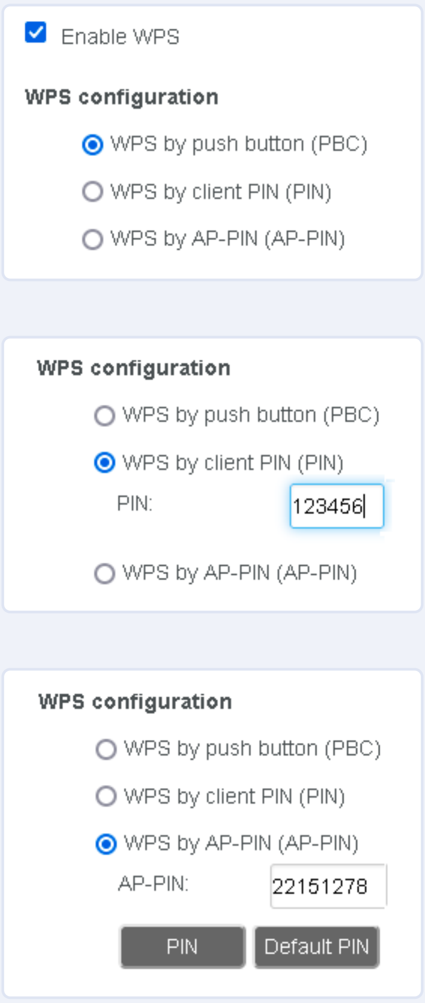
In this mode, the WeMos checks the sanity of the external EEPROM, in my case a 24LC64 with 8 KB on board. **Figure 4** shows the output of this check in Serial Monitor. I didn't involve this external device any further in the project since utilization of the internal EEPROM is minimal. The reader may refer to the I²C EEPROM code [3] and library [4] for integration in the project.

Mode 7: Direct Connection to WLAN (WPS)

WPS is a feature supported by modern APs that simplifies the process of connecting wireless devices to a WLAN. Rather than providing an MCU with the AP credentials, e.g. SSID and passphrase/pre-shared key, the MCU may provide the AP security pin or its own preset pin (see **Figure 5**). Alternatively, a WPS push button may be pressed on the AP for direct connection to the requesting MCU (see **Figure 6**). **Figure 7** shows the outcome of establishing a connection using WPS.

In this project, I confine the WPS connection to the AP push button alternative as it makes coding simpler and more generic to accommodate any AP within reach, as follows:

```
bool wpsSuccess = WiFi.beginWPSConfig();
if (wpsSuccess) {
    String newSSID = WiFi.SSID();
    if (newSSID.length() == 0)  { wpsSuccess = false; }
}
```



[✓] Enable WPS

WPS configuration

[✓ WPS by push button (PBC)]
WPS by client PIN (PIN)
WPS by AP-PIN (AP-PIN)

WPS configuration

WPS by push button (PBC)
[✓ WPS by client PIN (PIN)]
PIN: 123456
WPS by AP-PIN (AP-PIN)

WPS configuration

WPS by push button (PBC)
WPS by client PIN (PIN)
[✓ WPS by AP-PIN (AP-PIN)]
AP-PIN: 22151278

Figure 5: AP configurations for WPS alternatives.

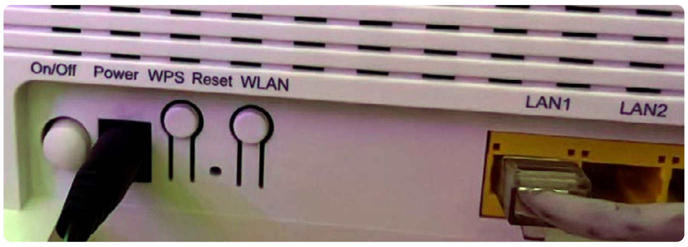
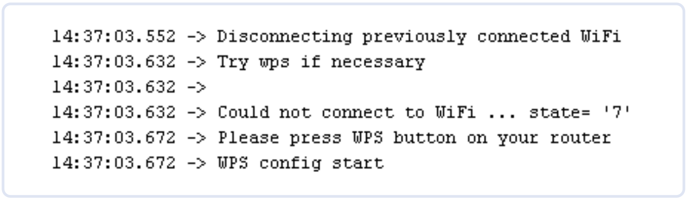


Figure 6: WPS button and indicator of a typical AP.



```
14:37:03.552 -> Disconnecting previously connected WiFi
14:37:03.632 -> Try wps if necessary
14:37:03.632 ->
14:37:03.632 -> Could not connect to WiFi ... state= '7'
14:37:03.672 -> Please press WPS button on your router
14:37:03.672 -> WPS config start
```

Figure 7: Serial Monitor messages — Mode 7 WPS setup.

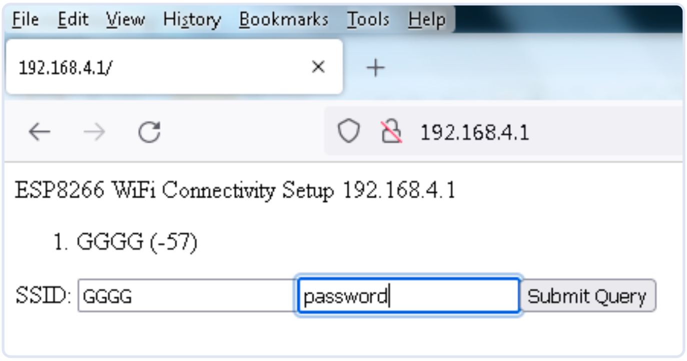
The WeMos board needs to be in close vicinity to the AP of choice for registering the AP parameters in the module's EEPROM. Reconnecting the module to the same AP no longer requires pressing the WPS button anymore, as we enable the module to retrieve the stored AP parameters for reuse, like this:

```
WiFi.mode(WIFI_STA);
WiFi.begin(WiFi.SSID().c_str(),WiFi.psk().c_str());
```

Note that the ESP8266 module that forms the heart of the WeMos board automatically stores the recent successful networking credentials in the internal EEPROM, so we need to place our housekeeping structure, depicted in Table 3, elsewhere in the EEPROM so as to avoid conflict with the MCU's other mechanisms.

Mode 8: Interactive WLAN Setup

In this mode, the WeMos checks the sanity of the credentials in EEPROM first. If the MCU succeeds in connecting to an AP, this mode ends and the `loop()` function takes over. If not, then the MCU enters the interactive mode, in which it scans the WLANs in its vicinity and launches a web server with IP address **192.168.4.1** to display the detected AP SSIDs. The user needs to set their computer's WLAN adapter to the fixed IP settings for the WeMos private network, since the WeMos will provide no DHCP service. A webpage close to that shown in **Figure 8** will show the scan results, and enable the user to choose the preferred network. The user then clicks on the *Submit Query* button to save the WLAN credentials in EEPROM and establish a connection to the selected WLAN.



ESP8266 WiFi Connectivity Setup 192.168.4.1

1. GGGG (-57)

SSID: GGGG password

Figure 8: Mode 8, the interactive WLAN selection.

EEPROMs can make a difference when networking wireless MCUs. Rather than hard-coding WLAN credentials in the MCU code, accessibility to either an external or an internal MCU's EEPROM has been used in this article to dynamically keep and update the credentials of multiple APs for feasible connections. The source code accompanying the article is modular and well-structured, easy to understand, and provides reusable code for handling EEPROMs, WLANs, and for communicating with MCU through the web.

230268-01

About the Author

Gamal Labib has been an enthusiast of embedded systems for two decades and is currently a mentor (at codementor.io). He holds an MEng and a PhD in IT. Besides writing for technical magazines, he is a visiting associate professor at Egyptian universities and a certified IT consultant.

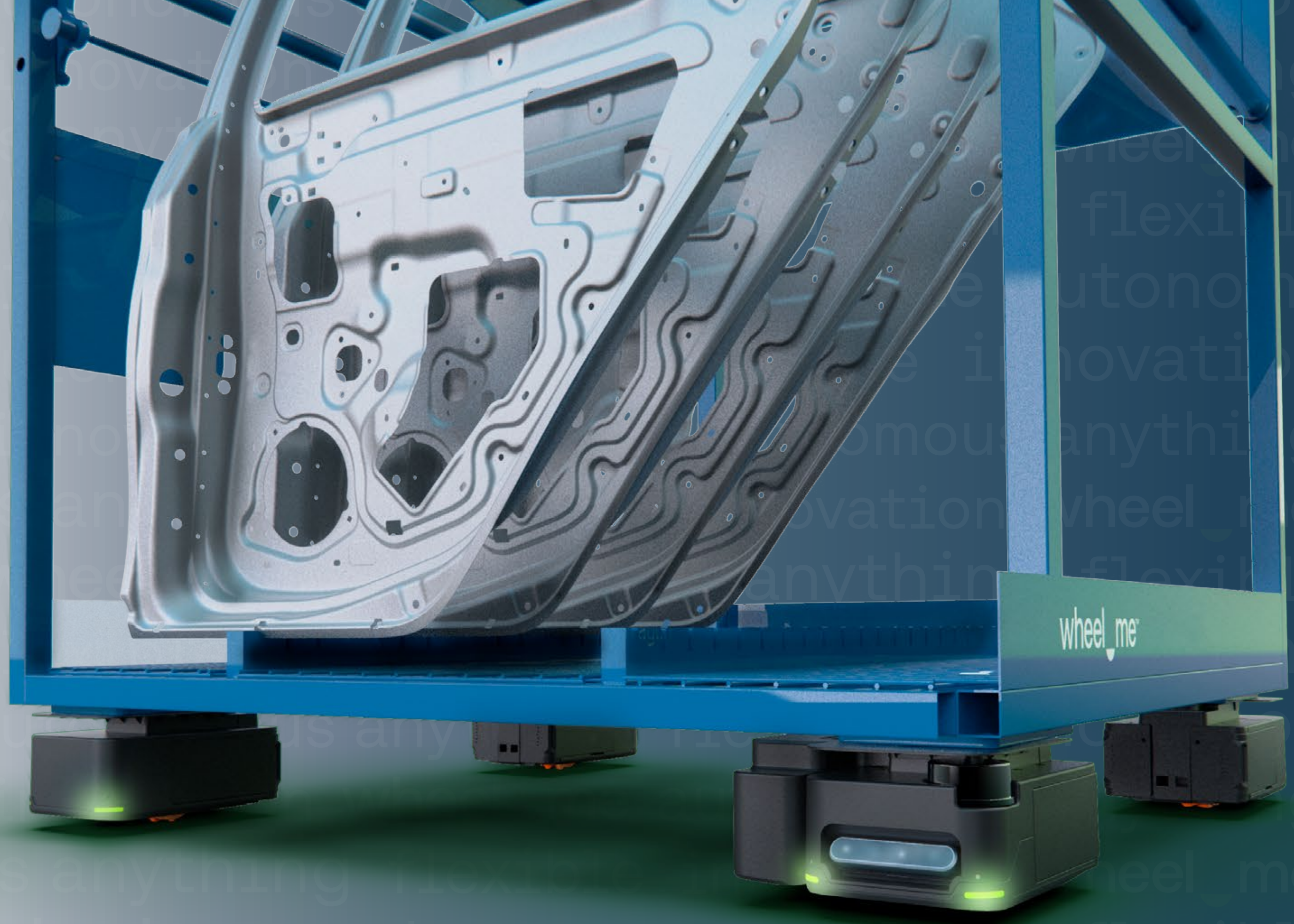
Questions or Comments?

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

Home Appliance Hack-and-IoT Guidebook
Affordable solutions with the ESP8266 and 3D printing
Hans Henrik Skovgaard
elektor

WEB LINKS

[1] Software download: https://elektormagazine.com/230268-01
[2] Using the EEPROM with the ESP8266: https://aranacorp.com/en/using-the-eeprom-with-the-esp8266
[3] Arduino with an I2C EEPROM: https://playground.arduino.cc/Code/I2CEEPROM
[4] Library for I2C EEPROM: https://github.com/RobTillaart/I2C_EEPROM



Genius by wheel_me

Enable peak efficiency in intralogistics with wheel.me's Genius: Four adaptable robot wheels enabling autonomous transportation of goods at a competitive price.

- Smart navigation
- Omnidirectional movement
- State of the art sensor technology
- Remote access via App
- Flexibility to adjust your payload
- In-process charging

wheel_me
more details at
www.wheel.me

€5,000 up for grabs!

Join the STM32 Wireless Innovation Design Contest

By Clemens Valens (Elektor)

STM32 Wireless Innovation Design Contest gives you the opportunity to create and develop exciting wireless applications with powerful products supported by a rich ecosystem including development and evaluation boards from STMicroelectronics. Use them for whatever you think is interesting — and in any way you like! IoT, robotics, gaming, home automation, test and measurement, and AI are just a few of the possible application fields. It's all up to you. Release your creativity, have fun and win! A total of €5,000 is up for grabs!

NUCLEO-WBA52CG

Bluetooth applications have gained a lot of popularity over the past years. At the time of writing, the Bluetooth SIG website expects that 7.6 billion Bluetooth-enabled devices will ship annually by 2027. The standard itself keeps evolving, and version 5 introduced IoT support. Currently, the Bluetooth Core Specification is at version 5.4.

The STM32WBA52CG is an RF/SoC microcontroller supporting Bluetooth LE 5.3. This is an ultra-low-power ARM Cortex-M33 device running at up to 100 MHz. It has 1 MB of Flash memory and 128 KB of SRAM. The MCU implements a single-precision floating-point unit (FPU), plus a full set of DSP instructions. It has a memory protection unit (MPU) and it features Trust Zone. The device integrates a 2.4 GHz transceiver supporting Bluetooth Low Energy and proprietary protocols. The microcontroller is mounted inside a metal can, so you can't see it.

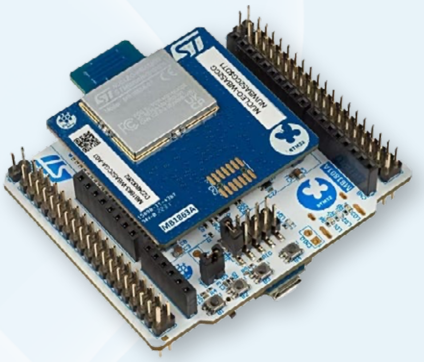
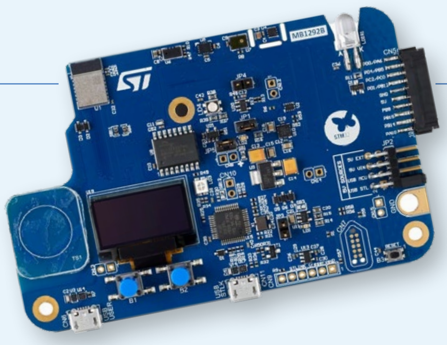


Figure 1: The NUCLEO-WBA52CG features an Arm Cortex-M33 with Trust Zone and Bluetooth LE 5.3.

The STM32WBA52CG is soldered to a breakout board that, in turn, is plugged onto another breakout board. This board is equipped with Arduino and ST Morpho-compatible extension headers, configuration jumpers, push



◄ *Figure 2: The STM32WB5MM-DK Discovery kit features many sensors and a small OLED display.*

buttons and LEDs, as well as a voltage regulator. An STLINK-V3 debugger/programmer module sits on the underside of the board. Although intended for making application development for the WBA52CG module easy, you might be tempted to desolder it and use it as a stand-alone STLINK-V3 debugger/programmer pod instead.

The NUCLEO-WBA52CG board comes preprogrammed with a demo application that can communicate with the ST BLE Sensor app on a smartphone. The app shows the status of pushbutton B1, and the app lets you switch on and off an LED on the board. This is nice, of course, but we are certain that you can do better. The board has many more application possibilities. The MCU's powerful security features allow for sensitive and secure IoT applications.

More information: ***https://st.com/en/evaluation-tools/nucleo-wba52cg.html***

STM32WB5MM-DK

The STM32WB5MM-DK Discovery kit is a demonstration and development platform for the STMicroelectronics STM32W5MMG module. This dual-core 32-bit Arm Cortex-M4/M0+ device integrates an ultra-low-power radio compliant with Bluetooth Low Energy (BLE) 5.2, 802.15.4 with Zigbee, Thread, and proprietary protocols.

The strangely-shaped board comes with many peripherals, such as an 0.96-inch 128×64 OLED display, a temperature sensor, an accelerometer/gyroscope sensor, Time-of-Flight (ToF), and gesture-detection sensor. It also has a digital microphone, an RGB LED and an infrared LED, two user push buttons and a touch key button. For applications that need storage space, there is a 128-Mbit Quad-SPI NOR Flash memory chip. An STMod+ and Arduino-compatible expansion connectors let you connect other devices to the board.

One might think that the STM32 processor sitting in the middle of all these circuits is the board's main processor, but that's not the case. It is the integrated ST-LINK/V2-1 that provides embedded in-circuit debugging and programming facilities and a USB-to-serial bridge. The main MCU is hiding behind the tiny metal can on the board's top-left corner.

The STM32WB5MM-DK Discovery kit comes preprogrammed with a Bluetooth audio application. It sends audio data captured with its digital microphone to the connected app on a smartphone. The app can send audio data back to the board. When running the demo in full-duplex mode, make sure to keep the phone or tablet at some distance from the board to avoid audio feedback (a.k.a. the Larsen effect) blowing your eardrums.

Because the dev kit has so many peripherals, there's a lot that you can do with it without having to add anything else. It is great for IoT and home automation applications, but, with a bit of creativity, you can easily come up with other exciting use cases.

More information: ***https://st.com/en/evaluation-tools/stm32wb5mm-dk.html***

NUCLEO-WL55JC

The NUCLEO-WL55JC board is an evaluation board for the STM32WL-series of microcontrollers, and, in particular, the STM32WL55. This so-called sub-GHz wireless microcontroller is based on a dual-core 32-bit ARM Cortex-M4/M0+ with a clock frequency of 48 MHz. It features ultra-low power consumption, an integrated RF transceiver with a 150 MHz to 960 MHz frequency range, 256 KB of Flash memory and 64 KB of SRAM.

The RF transceiver inside the MCU supports LoRa, (G)FSK, (G)MSK, and BPSK modulations. Being a fully open wireless system-on-chip, it is compatible with standardized as well as proprietary protocols such as LoRaWAN, Sigfox, wM-Bus, and more. The transmitter has a high-output-power mode, programmable up to +22 dBm, and a low-output-power mode, programmable up to +15 dBm. In Europe, the uplink transmission power is limited to 14 dBm (25 mW), so it is important to check regulations for the area where you are planning to use this board.

The board the MCU is mounted on is equipped with extension headers (Arduino and ST morpho compatible), configuration jumpers, pushbuttons, and LEDs, and a voltage regulator. There is also an on-board STLINK-V3

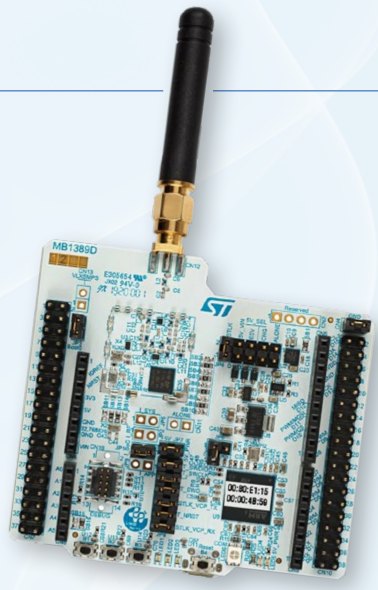


Figure 3: The NUCLEO-WL55JC is compatible with standardized as well as proprietary protocols such as LoRaWAN, Sigfox, wM-Bus and more.

debugger/programmer to facilitate application development. An SMA antenna is also included.

The NUCLEO-WL55JC board comes preprogrammed with a sensor data concentrator demo application. The WL55JC board can be turned into a compatible sensor node too, if you upload another firmware example from the STM32CubeWL library package. This is available from the product page on ST's website.

The board has, of course, many more application possibilities. One that immediately springs to mind is a LoRaWAN end node. Instructions explaining how to do this can be found by searching the ST website.

The NUCLEO-WL55JC board comes in two versions: the WL55JC1 is for use in the 865-928 MHz band, whereas the WL55JC2 is for the 433-510 MHz band.

More information: *https://st.com/en/evaluation-tools/nucleo-wl55jc.html*

230442-01

Libraries and Development Toolchains

All STM32 products are supported by the STM32Cube platform. STM32Cube is ST's original initiative to simplify and ease the life of the developer by reducing development effort, time, and cost. It provides developers with a hardware abstraction layer (HAL) and the low-layer (LL) APIs, a consistent set of middleware components, and many application examples that can be easily reused for custom application development.

STM32Cube includes STM32CubeMX, a graphical software configuration tool that uses graphical wizards to help the developer generate C initialization code.

A suitable toolchain for developing applications for the three boards presented in this article (and many more) is, of course, STM32CubeIDE from STMicroelectronics themselves, which is free of charge. Keil's MDK-ARM and IAR's Embedded Workbench can be used too, but at the developer's expense.
https://st.com/en/development-tools/stm32cubeide.html

The stm32duino boards package for the Arduino IDE supports the STM32WB5MM-DK board. It also knows about the WL55JC, but without LoRa support. The WBA52CG board is not (yet) supported.
https://github.com/stm32duino

Both the STM32WB5MM-DK and WL55JC boards are also compatible with ARM's mbed OS, so that is another avenue to explore.
https://os.mbed.com/platforms/DISCO-WB5MMG/
https://os.mbed.com/platforms/ST-Nucleo-WL55JC/

STM32

Innovate to win!
€5,000 in prizes!

How to Participate?

To inspire and incentivize participants, STMicroelectronics is offering the following prizes:

1st prize: €2,500
2nd prize: €1,500
3rd prize: €1,000

For details on how to participate in the STM32 Wireless Innovation Design Contest, the timeline, and the exact terms and conditions, please refer to the contest entry webpage at

elektormagazine.com/st-contest

2023: An AI Odyssey

Getting Started with ChatGPT's Code Interpreter

By Brian Tristam Williams (Elektor)

ChatGPT's capabilities have reached new heights with the release of the ChatGPT Code interpreter plugin. Code Interpreter is now available for all ChatGPT Plus users. But, what is it, and what does it do? Let's take a closer look at this unique and useful ChatGPT plugin.

OpenAI's ChatGPT is revolutionizing our interaction with artificial intelligence. With the recent addition of the Code Interpreter, it has transformed into more than just a chatbot — it's now a powerful assistant for developers, data scientists, and coding enthusiasts.

OpenAI's mission to push the boundaries of AI capabilities has led to groundbreaking developments, such as the various ChatGPT [1] models, significant leaps in natural language processing. The recent upgrade, GPT-4, builds upon this impressive foundation, but it's the introduction of the Code Interpreter plugin [2] that truly sets it apart.

Code Interpreter

We have to note that the term "interpreter" is ambiguous here. Even the free version, GPT-3.5, is capable of interpreting code (meaning figuring out what code does) and providing meaningful feedback in natural language. That's not what they mean here. This is an interpreter in the sense that it can actually *run* our Python code as an interpreted language, in its own sandboxed and firewalled environment, along with its own virtual disk space. Just as when you used "immediate mode" in gold old-fashioned traditional BASIC, your Python session remains persistent for the chat session.

Code Interpreter is a game-changer. It goes beyond just executing Python code — it interprets, debugs, and even converts files between formats. Users can input mathematical problems, and Interpreter will solve them. It can also take data in various file formats, analyze it, and visualize it, meaning that it isn't just limited to straightforward code execution. It can generate data visualizations at a speed that has left long-time data analysts astounded.

Use Cases

Code Interpreter's applications are myriad, and many are still to be discovered. Some of what we've already been trying:

- **Code testing**: Users can test code snippets in real-time, ensuring they work as intended.
- **Collaboration**: Teams can collaborate on coding projects directly within the chat.
- **Learning tool**: It's an excellent resource for students learning Python, offering interactive coding sessions. This is huge for me — it's like having your own private tutor on the other side of a chat window.
- **Experimentation**: Developers can try out different programming approaches without leaving the chat.
- **Solving mathematical problems**: Whether it's high school algebra or differential calculus, the right prompts will get you your solution and walk you through how to get there.
- **Format conversion**: Take data in various file formats and analyze it, clean it up, and visualize it.

Getting Started

While there is a free version of ChatGPT, you need to be a Plus subscriber to use plugins such as Code Interpreter. At the time of writing, it's in Beta, so you'll have to enable the feature. On a desktop browser, get there by clicking on your profile image on the bottom-left, followed by the *Settings* cog (**Figure 1**). Then, in the dialog that follows, click on *Beta features*, and make sure that both the *Plugins* and the *Code interpreter* toggles are on (**Figure 2**).

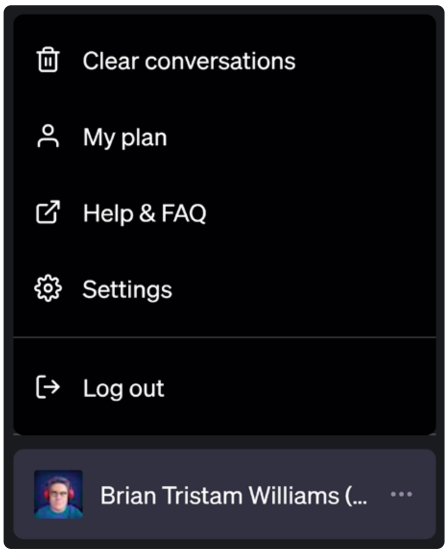
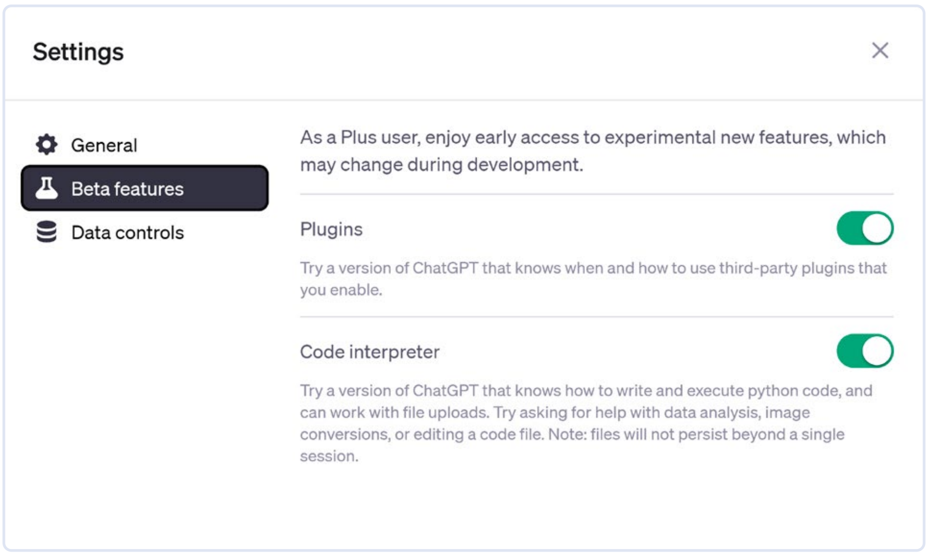


Figure 1: How to get to Settings in ChatGPT.


Settings

As a Plus user, enjoy early access to experimental new features, which may change during development.

Plugins

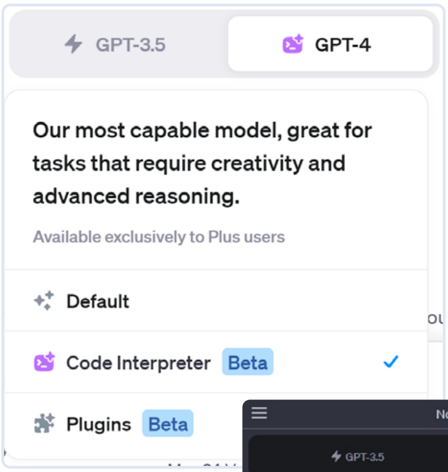
Try a version of ChatGPT that knows when and how to use third-party plugins that you enable.

Code interpreter

Try a version of ChatGPT that knows how to write and execute python code, and can work with file uploads. Try asking for help with data analysis, image conversions, or editing a code file. Note: files will not persist beyond a single session.

Figure 2: Make sure that Plugins and Code interpreter are enabled.

Close the dialog, and start a *New chat*. At the top of the chat window, you'll be given the option of selecting between *GPT-3.5* and *GPT-4*. The latter has a drop-down menu, which will allow you to select between the *Default* mode, *Code Interpreter*, and other plugins. Click on *Code Interpreter* (**Figure 3**).



Our most capable model, great for tasks that require creativity and advanced reasoning.

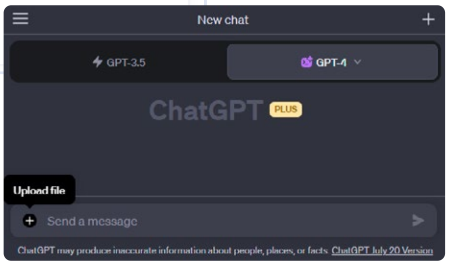
Available exclusively to Plus users

Figure 3: Getting to Code Interpreter in GPT-4.

Trying It Out

Besides being a Python run-time environment, Code Interpreter is capable of parsing and analyzing a wide variety of files, from PDFs, to Excel spreadsheets, to comma-separated values (CSV).

With so many use cases, we're overwhelmed for choice, but one such toe-wetting experiment I did was to analyze a complete Elektor book that I edited last year. These are the basic steps I followed, which you can use to get started finding insights from your own data:



ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT July 20 Version

Figure 4: Click on the "plus" sign to the left of the input field to upload your document.

Upload its PDF by clicking on the "plus" sign to the left of the input field (**Figure 4**), navigating to the file and select it for uploading. The book I used, *Develop Your Own Bluetooth Low Energy Applications for Raspberry Pi, ESP32, and nRF52 with Python, Arduino, and Zephyr*, by Koen Vervloesem (see the **Related Products** box), is a 41-megabyte file consisting of 258 pages, so I expected to encounter challenges from ChatGPT.

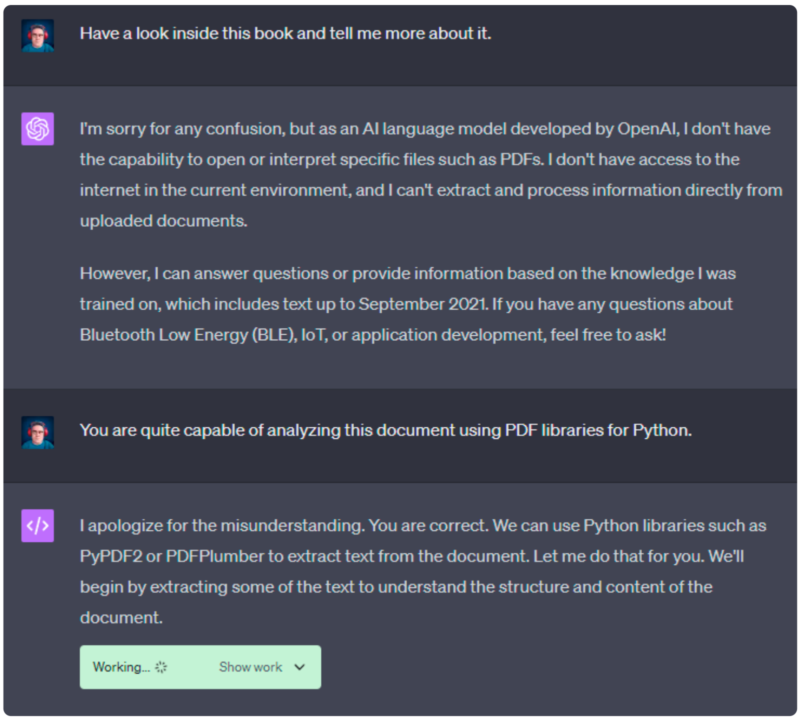
First, I uploaded the file in its entirety, and asked ChatGPT to tell me about it (**Figure 5**). The rest of the chat between me (BTW) and Code Interpreter (GPT) is too verbose to reprint here, but here's a summary and some takeaways:

BTW: Tell me about this book.
GPT: It has 258 pages and its basic metadata tells me it's about BLE. Must I look inside and tell you more?
BTW: Yes, please.
GPT: Error! The way I'm parsing this in Python is not working. Read the book's table of contents yourself.

When you hit this wall, you may be tempted to give up, but you can often coax the model into using a workaround.

BTW: Can you use another method to parse the PDF?
GPT: Sure, let's try the *PDFMiner* library instead of *PyPDF2*. The first 2,000 characters tell me the book is about.
BTW: Read the entire book, not just the first 2,000 characters.
GPT, trying to get out of work, just like a human: This will be difficult and resource-intensive. Let me tell you more basics about the book.
BTW: Read the entire book — I need to ask specific questions.
GPT: Fine.

After convincing it to actually look at the full document, there were still some niggles that the environment had to go through, due to the size and complexity. The code execution environment crashed and had to be reset (by ChatGPT automatically), libraries were misplaced (but not by me), and the entire book had to be uploaded again. Eventually, I was able to glean some insights:


Have a look inside this book and tell me more about it.

I'm sorry for any confusion, but as an AI language model developed by OpenAI, I don't have the capability to open or interpret specific files such as PDFs. I don't have access to the internet in the current environment, and I can't extract and process information directly from uploaded documents.

However, I can answer questions or provide information based on the knowledge I was trained on, which includes text up to September 2021. If you have any questions about Bluetooth Low Energy (BLE), IoT, or application development, feel free to ask!

You are quite capable of analyzing this document using PDF libraries for Python.

I apologize for the misunderstanding. You are correct. We can use Python libraries such as PyPDF2 or PDFPlumber to extract text from the document. Let me do that for you. We'll begin by extracting some of the text to understand the structure and content of the document.

Figure 5: The file is uploaded. Let's get cracking on the analysis, even if we have to remind the computer of its capabilities.

- The three most-mentioned accessory/microcontroller boards were the nRF52, the ESP32, and the Raspberry Pi, with 149, 136, and 30 mentions, respectively.
- Helpful subjects to have a grasp of before beginning this book: A basic understanding of the C and Python languages; some experience interacting with microcontrollers; general understanding of wireless communication protocols; using the Linux command line; basic networking concepts.
- The book contains 121 URLs, and Code Interpreter was able to output the entire list for me. This a boon for both editor and publisher, who need to check if links work, or have expired, or had typos. So, I had Code Interpreter write me a Python script to check all of these links and return the HTTP status codes.

It turns out that a couple of websites, including big ones, have restructured already (this book is only a year old), breaking some of those links. Furthermore, some of the links in the ebook are misinterpreted by Adobe PDF due to formatting (line breaks, etc.), which does not affect the print edition that we poured over. This leads me to useful, actionable, ideas:

- For each book or magazine publication, an automated script could run daily, validating all the links, checking for "404 not found" errors or suspicious new redirects (as may happen if a domain expires and is snapped up by a third party). Alert the publisher where necessary.
- While it was never previously easy to determine how much of a problem newly-broken links are once an item is published, an automated script could pass the data to Code Interpreter in a CSV file for automated analysis, and we could gain a far better idea of how often links become invalid versus their age, for example.
- If links change so often that it is a concern, consider using a branded link shortener, which can be updated whenever manufacturer X or software publisher Y decides to move their page somewhere else, or a smaller website gets switched off and we have to rely on a third-party archive such as *archive.org*.

The Sky Is the Limit

The basic exploration above barely scratches the surface of what's possible and the potential it has. In demos shown on OpenAI's blog posts [3], we've seen the plugin performing a wide range of tasks excellently, interpreting large datasets, identifying trends, comparing variables, and creating plots. The plugin can even be used to create GIF animations from a short prompt.

I have gone on many meandering tours of many subjects, including a conversation about discrete mathematics, Amida-kuji, Hamiltonian cycles, the Towers of Hanoi, and gray code, with my trusty and patient guide, and it has even generated Python code to, in turn, generate visualizations of ideas I wanted to test [4].

Now that the internet has had more time to use the Code Interpreter plugin, we've seen even more uses come to light. Have you tried it? If so, let us know how it's helped make your life easier!

230181-B-01

Questions or Comments?

Have you found any amazing uses for AI in your field? Do you have questions or comments about this article? Let me know at brian.williams@elektor.com or contact editor@elektor.com.

WEB LINKS

[1] OpenAI's ChatGPT: https://chat.openai.com
[2] OpenAI announces Code Interpreter [Tweet]: https://twitter.com/OpenAI/status/1677015057316872192
[3] OpenAI blog post introducing Code Interpreter: https://openai.com/blog/chatgpt-plugins#code-interpreter
[4] Chat Log: Discussing and Visualizing Discrete Mathematics: https://tinyurl.com/discretegptchat

LoRa, a Swiss Army Knife (1)

The LoRa Protocol and Its Advantages

By Gilles Brocard (France)

This article describes the practical application of the LoRa transmission protocol using Ebyte modules based on Semtech's LLCC68 transceiver, with simple resources and a limited budget. This first part presents the LoRa protocol and its advantages.

Today, transmitting and receiving data via radio waves has become so commonplace that we don't really pay attention to it. 3G, 4G, and now 5G enable us to transmit and receive information at high speed, but over relatively short distances, mostly a few hundred meters. Wi-Fi and Bluetooth is even less. But, some fields have different needs. Let's take the example of a farmer who wants to send data to a central collection system for animal counting, local temperature (frost monitoring) or any other activity requiring the use of measurements supplied by a sensor several kilometers away. To meet these needs, solutions have been developed with some interesting innovations. You can find more on LoRa in several Elektor articles, for example in [1] and [2].

In essence, LoRa is a radio signal transmission protocol that uses a "chirped" multi-symbol format to transmit data. LoRa chips operate in the ISM bands and convert radio frequencies into data. For this, they use LoRa technology, which is a low-level physical layer. We'll come back to these terms in a moment.

The use of wired modules is a simple solution for low-speed data transmission between two distant points. The Semtech company [3] owns the LoRa patents, and has developed a whole range of LoRa transceiver circuits.

Ready-to-Use Modules

Currently, the best-performing circuit is the LLCC68, but it's not easy to use. First of all, it's a very small SMD — 4 mm square for 24 pins — and therefore very difficult to solder. In addition, it needs to be surrounded by RF components compatible with the gigahertz frequencies, all on a printed circuit board adapted to the constraints of these high frequencies. Chinese company Ebyte, specialized in the RF field, has developed a whole family of small modules integrating Semtech's circuits.

The Ebyte range comes in many models. Models E220-900T22D (**Figure 1**) and E220-900T30D (**Figure 2**) contain an LLCC68 with a microcontroller located between the single access, the UART communication port, and the LLCC68. However, this microcontroller prevents direct communication with the LLCC68, which considerably limits the configuration possibilities of the LLCC68. **Figure 3** shows, with the shield removed, the LLCC68, the microcontroller (ARM CX32L003),

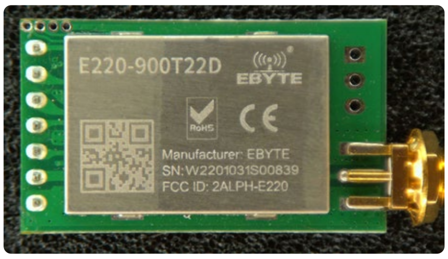


Figure 1: Ebyte's E220-900T22D module.

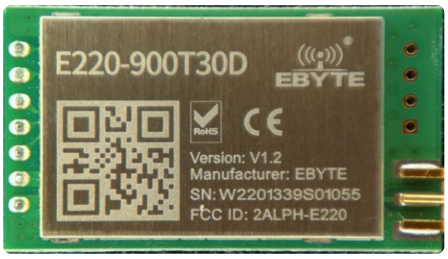


Figure 2: The E220-900T30D module incorporates an 8 dB power amplifier, increasing transmission power to 30 dBm or 1 W.

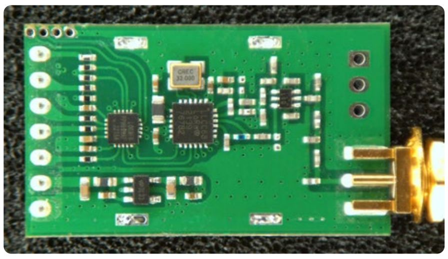


Figure 3: The E220-900T22D module without its shielding.

Figure 4: Modules ending in 'S' offer IPEX / U.FL antenna output.

a regulator, and an RX/TX RF output switcher. All other components are resistors or capacitors and HF chokes, as well as a 32 MHz oscillator, controlled by a crystal.

Modules with the '30' radical also incorporate an 8 dB power amplifier, raising the transmitting power to 30 dBm or 1 W. They are slightly larger in size, and consume considerably more power at full output (750 mA instead of 150 mA for Model 22). This power reserve can be invaluable in compensating for losses of all kinds caused by HF cables and connections in the gigahertz range.

The two aforementioned models have an antenna output with a female SMA connector, but these modules are also available with an IPEX / U.FL antenna output; the reference ends with 'S' instead of 'D' (**Figure 4**).

Without Integrated Microcontroller

The module we're going to use, the E220-900M30S, is of particular interest to us, as it has no microcontroller on the control lines. In fact, the LLCC68 is directly accessible via an SPI connection (the model is identified by the letter "M"). This means that all LLCC68 parameters can be accessed without restriction, which is essential if we are to exploit the full benefits of LoRa.

This model, like its predecessors, is available for different frequency bands, from 150 to 930 MHz. We have chosen the 900 model, which covers the 868 MHz band authorized in France. This module can easily be purchased from major distributors for between 3 and 10 euro, depending on model, power and supplier.

We're combining this module, along with a few peripheral components, with a microcontroller whose programming we've mastered and for which we can create applications. In this way, we'll be able to make all kinds of LoRa connections and interesting applications. This module will become your LoRa "Swiss Army knife."

A Little More About LoRa

The LoRa protocol enables links over relatively long distances (several kilometers) at very low power levels (tens to hundreds of milliwatts), but with the drawback that it's only at low data rates. As soon as you increase the (configurable) data rate, the range is reduced. This means that LoRa is very effective for low-bitrate applications such as gate control transmission, very long-range remote control, and reception of remote sensors that don't require frequent updating.

We chose the 868 MHz frequency because it's available in Europe without the congestion of the 433 MHz band. In addition, it's the frequency chosen for the LoRaWAN protocol. It's also regulated, just like all ISM bands (Industrial, Scientific and Medical). The advantage of ISM bands is that as long as you comply with European regulations, you can transmit radio signals without any prior declaration.

Radio Spectrum Sharing

Radio spectrum is a common resource, so we need to share it so that we don't disturb others with our transmissions, and aren't disturbed by theirs. LoRa uses three ways of sharing a frequency band:

Frequency sharing: The available frequency band is divided into contiguous channels more or less evenly spaced (**Figure 5**).

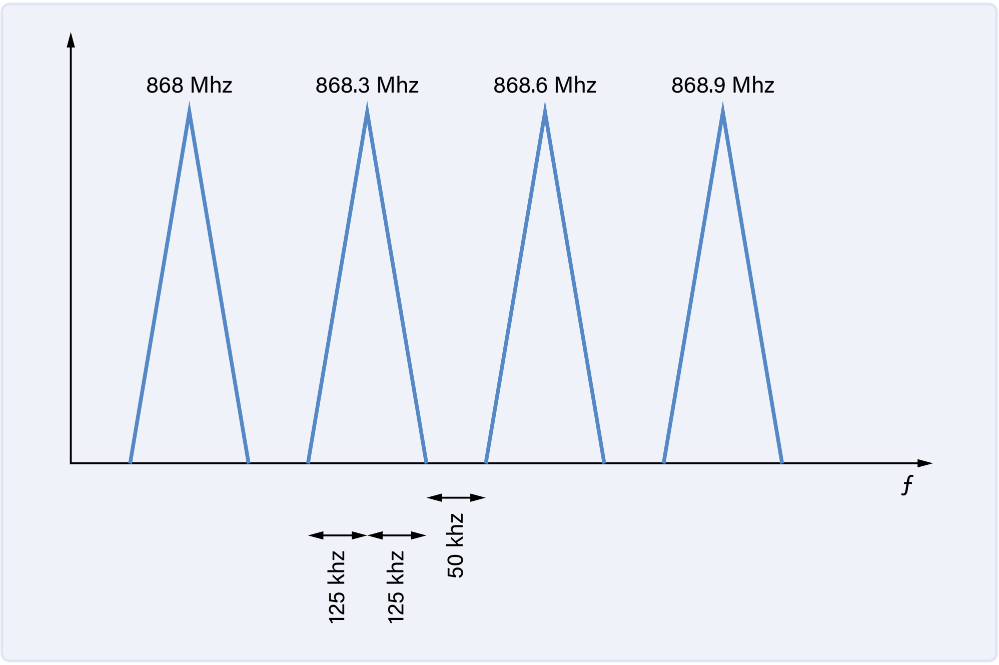


Figure 5: LoRa modulation uses a configurable bandwidth. This example shows a BW bandwidth of 250 kHz, i.e. 125 kHz on either side of the center frequency. The frequency deviation will therefore range from 868.175 MHz to 868.425 MHz for the second frequency of 868.3 MHz.

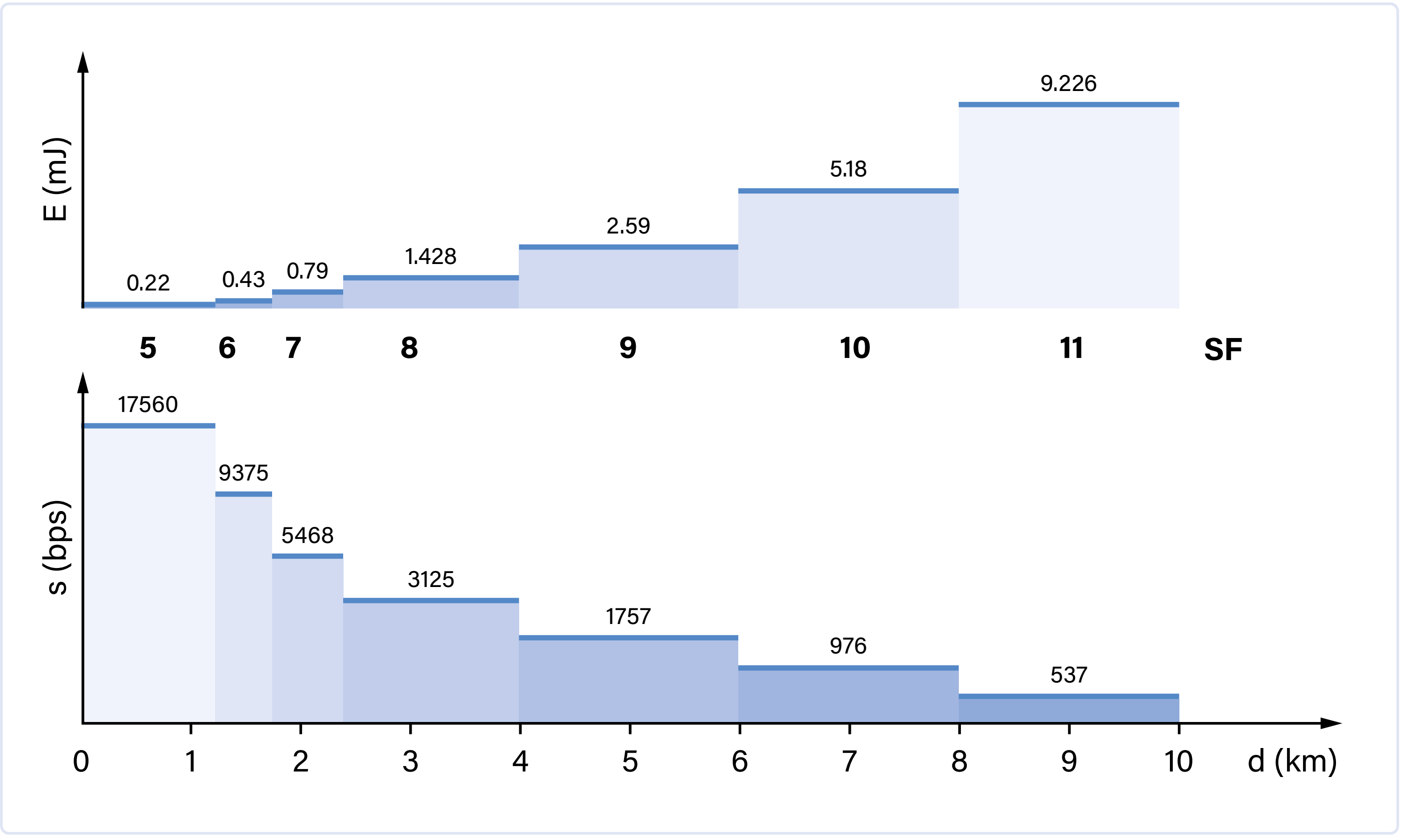


Figure 6: The influence of the ***spreading factor*** *(SF, in bold) on transmission speed and range (bottom), and consumption (top).*

Time-sharing: With LoRa, transmissions can and must be of short duration and spaced out over time. In this case, the same band can be used successively by several devices. Although this technique is prone to collisions, it is widely used in LoRa modulation. One way of limiting collisions is to monitor the frequency band and only transmit when the band is not in use. The LLCC68 makes it easy to automate this procedure (see RSSI below).

Spread-spectrum sharing: This mode is specific to LoRa, which digitizes the message using chirps (frequency deviations) — hence the name of this modulation (CSS for chirp spread spectrum). The LLCC68 lets you choose between seven spreading codes (spreading factor — "SF" in the datasheet). The values, from 5 to 11, depend on the bandwidth chosen (**Figure 6**). The SF is one of the most important parameters in LoRa modulation. In particular, it determines the range, the transmission speed, and the power consumption during transmission. Bandwidth (BW) and CR (code rate / coding redundancy) are the other two important LoRa parameters.

Link Budget

The link budget is the difference between the maximum transmit power, in this case 30 dBm, and the minimum input level required to receive and decode a message (sensitivity), in this case, 129 dBm. It is valid for the most favorable setting with BW = 125 kHz and with the receiver preamplifier activated. The maximum link budget is therefore 30 + 129 = 159 dBm. European regulations limit transmit power to 14 dBm, which, in our example, produces a link budget of 14 + 129 = 143 dB.

You may be surprised that we've chosen a module that can transmit at a power of 30 dBm (1 W), whereas European regulations only allow up to 14 dBm (25 mW). In fact, this 14 dBm is the power radiated by the antenna. But, between the module's HF output and the antenna, there are many causes of attenuation, which will reduce the power reaching the antenna. If you're lucky enough to still have an excess of transmit power, and you can measure it, the LoRa module's settings will enable you to reduce it.

A Practical Example

- Emission: 14 dBm power (limited by your settings).
- Transmitting antenna gain: +2.15 dBi (for a standard 86 mm quarter-wave ground plane antenna).
- Maximum signal attenuation over a distance of 11 km through the air: –111.2 dB (see formula for calculating this rating below; negative value indicates attenuation).
- Receiving antenna gain: +2.15 dBi (for an identical 86 mm standard ground plane quarter-wave antenna).
- Receiver sensitivity (LLCC68 for BW = 125 kHz, SF = 7), max receive gain: –124 dBm (data sheet, page 19, first line of *Sensitivity LoRa* paragraph).
- Add –5 dB for losses due to links (cables, connections, etc.) for the transmitter module and –5 dB for losses due to the receiver.

The link budget can be obtained simply by adding the values in dBm, dBi, and dB — that's the whole point of decibels:

14 + 2.15 – 111.2 + 2.15 – 5 – 5 = –102.9 dBm

This value is well above the threshold of 124 dBm, which is the LoRa reception threshold for this module (receiver sensitivity). We have a margin of 21.1 dBm between the two, which should enable us to receive a usable signal.

A LoRa specificity: the Spreading Factor (SF)

SF	Chirps	SNR	BW 125 kHz	BW 250 kHz	BW 500 kHz
5	32	−2.5	yes	yes	yes
6	64	−5	yes	yes	yes
7	128	−7.5	yes −124 dBm	yes −121 dBm	yes −117 dBm
8	256	−10	yes	yes	yes
9	512	−12.5	yes −129 dBm	yes	yes
10	1024	−15	no	yes −129 dBm	yes
11	2048	−17.5	no	no	yes −127 dBm

Table 1. The summary of measurements presented on page 19 of the LLCC68 data sheet concerning the Spreading Factor.

First column

The spreading factor SF depends on bandwidth BW. For example, for BW = 125 kHz, SF can have a value between 5 and 9; for BW = 250 kHz the value 10 can also be used; and for BW = 500 kHz the value 11.

Third column

The signal-to-noise ratio (SNR) as a function of the SF parameter value.

Don't be surprised to see negative SNRs, as the LoRa demodulator is capable of recovering a signal well below the RF noise level. For example, at BW = 500 kHz and SF = 11, the recoverable signal may be 17.5 dB below noise. This means that the signal can be 56 times weaker than the radio noise level...

Second column

The second column will curb our enthusiasm, as it gives the transmission coding size required to achieve this performance. Transmission time is directly proportional to these values, and so is power consumption. The module's instantaneous consumption remains constant; it simply occurs over a longer period of time.

Fourth, fifth, and sixth columns

For BW = 125 kHz and SF = 7, the LLCC68 is able to extract the payload from a received signal of -129 dBm (79 nV_{RMS}). This value is particularly low: The LLCC68 is extremely sensitive for the analog part and very efficient for the digital part applying the LoRa protocol!

Scope Evaluation

Disregarding ambient electromagnetic interference, which we can obviously never achieve in practice, it is possible to calculate a link's theoretical maximum distance. The formula for the distance is:

$$\text{distance} = \sqrt{\frac{10^{\left(\frac{\text{Link Budget}}{10}\right)}}{1755 \cdot \text{frequency}^2}}$$

Link budget is the link budget of the module used for our example, in compliance with the European standard is 143 dBm.
Frequency is the frequency of the module: 868 MHz.
Distance is the range in kilometers (excluding HF noise).

In compliance with European regulations, the maximum link budget of 143 dB with our E220-900M30S in 14 dBm, the calculation gives a range of 388 km. But let's stop dreaming, this theoretical distance will never be reached in practice. Nevertheless, under the right conditions, a maximum range of between 10 km and 20 km can be achieved, on sight and outside urban areas.

As an example, in April 2020, a distance of 832 km was reached by a LoRa module of performance quite comparable to LLCC68. However, the transmission and reception conditions were highly optimized. In fact, an antenna was attached to a weather balloon at an altitude of 38 km, and reception was via a LoRaWAN gateway located in the middle of a mountainous area, thus rather well isolated from HF interference.

Transmission of a Symbol

LoRa modulation can be visualized directly with SDR software and an associated SDR key, as shown in **Figure 7**.

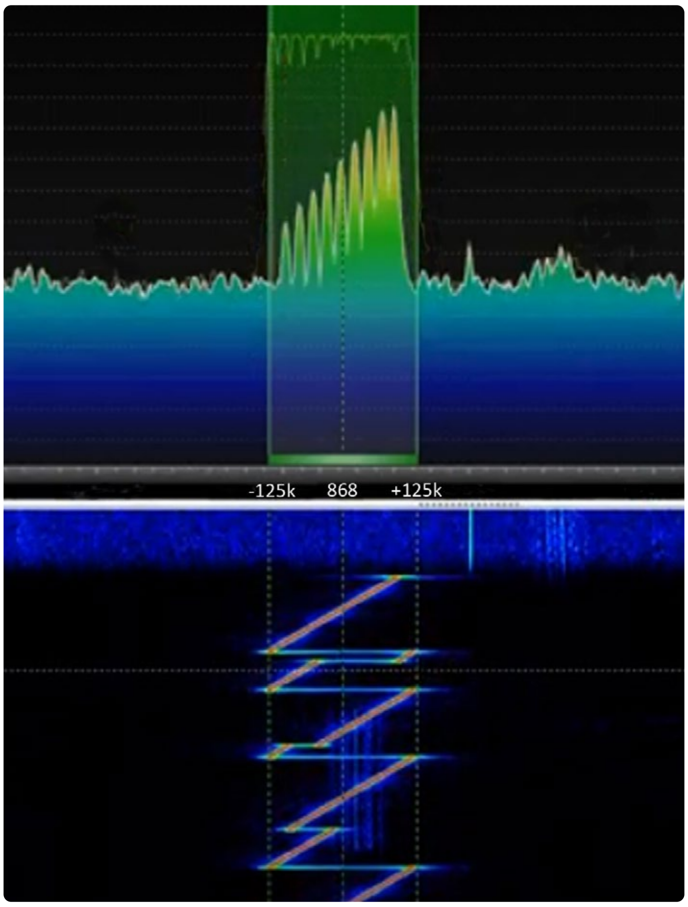


Figure 7: LoRa modulation can be viewed directly with SDR software and an associated SDR key. Here we see the modulation for a frequency of 868 MHz with a bandwidth of 250 kHz. The slowness of the LoRa transmission allows us to see the modulation of the symbol coding. At the top of the image, the vertical axis is graduated in average signal power (dBm) and the horizontal axis in frequency. In the lower part of the image, you can see the transmission sequence as a function of time, with the frequency variations characteristic of LoRa modulation.

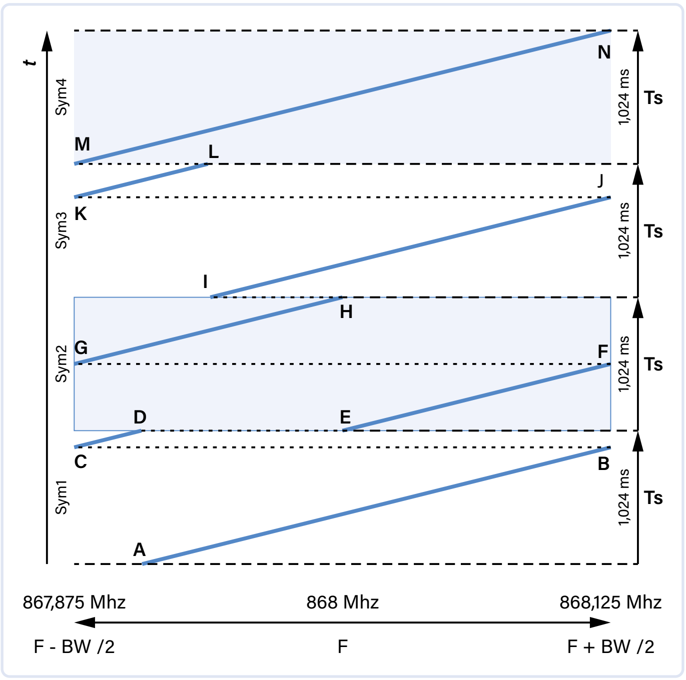


Figure 8: This chronogram shows the transmission of four symbols with SF = 8 and BW = 250 kHz. Warning: Time is represented by the vertical axis, frequency by the horizontal axis.

Figure 8 is a timeline showing the transmission of four LoRa symbols with SF = 8 and BW = 250 kHz. If our center frequency is 868 MHz, the frequency deviation comprises 2^8 = 256 increments to transit from the minimum frequency of 867.875 MHz to the maximum frequency of 868.125 MHz.

The coding of each symbol's information lies in the initial frequency value of the linear sweep of this frequency. For example, for symbol 1, the start frequency A, starts at 32/256 of the total frequency sweep, so the transmitted symbol encoding will be 32. For symbol 2, the start frequency E, starts at 128/256, so the symbol encoding will be 128. For symbol 3, the start frequency I, starts at 64/256, the symbol coding will be 64 and so on. For symbol 4, the start frequency M starts at 0/256 and the symbol encoding is 0. The values of the 4 transmitted symbols are therefore successively 32, 128, 64, and 0.

The frequency deviation is made in small increments, the resolution of which depends on the SF value. For SF = 8, $2^{SF} = 2^8$, i.e. 256 intervals; for SF = 11, there would be 2048. The duration of each increment Tc depends on the bandwidth. For BW = 250 kHz, they last 4 µs, for BW = 500 kHz their value is reduced to 2 µs and 8 µs for 125 kHz. This is why, with LoRa, the transmission time of a message is inversely proportional to BW and, on the other hand, it doubles for each increment of 1 of the value of SF.

This modulation process, borrowed from RADAR technologies, involves the transmission of symbols that are a linear frequency deviation of a sine wave around a center frequency. These frequency sweeps are called CHIRP (compressed high-intensity radiated pulse). For LoRa, only linearly increasing chirps are used for symbol transmission, such as those between points M and N. Linearly decreasing chirps are only used at the end of the preamble for synchronization.

The LoRa Package

The transmitted message is integrated into a physical frame called a packet. It consists of four parts (**Figure 9**), some of which are optional and/or configurable:

1. A preamble, always mandatory, but customizable. It's a succession of several (usually eight) increasing chirps and two decreasing chirps. This preamble is essential for the receiver to synchronize.

2. Header, optional. It is present by default if explicit mode is chosen (absent with implicit mode). It is always transmitted with a CR (code rate) of 4/8 to have the maximum chance of being received. It indicates:

   - Data size (*payload*).
   - Code rate (CR) used for the rest of the frame, between 4/5 and 4/8, but 4/5 is the most widely used. A code rate of m/n means that for every m bits of useful information, the encoder generates a total of n bits of data, of which m-n bits are redundant. Redundant bits enable the detection of transmission errors.
   - It specifies whether a CRC (error code) is present or not.

3. Data (payload). The maximum data packet size is 256 bytes. It depends on the value of the spreading factor and the buffer settings. The higher the SF value, the smaller the packet size. Maximum message duration is limited.

4. An optional error check (CRC). It checks whether the data received is complete and error-free.

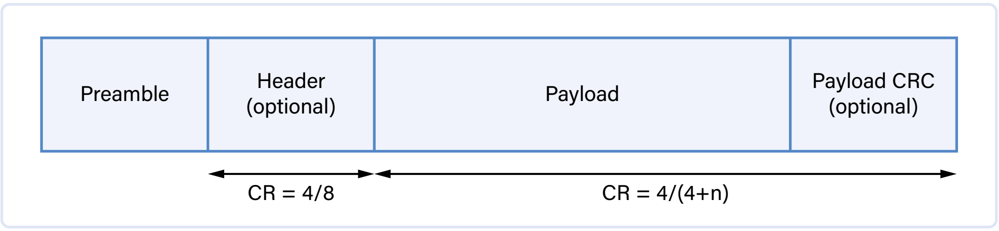
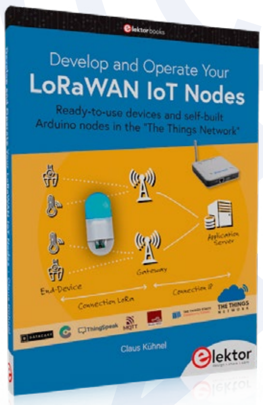


Figure 9: The LoRa package consists of four parts, some of which are optional and/or configurable.

In the rest of this article, we'll present the printed circuit board integrating the LoRa E220-900M30S module, controlled by an Arduino Nano. A C++ program allows the complete configuration of the LLCC68, and above all, to be used for transmitting and receiving messages. Range and autonomy settings are also detailed, with SF, BW, CR, LDRO, as well as the values returned by the receiver, RSSI, measured sensitivity, signal-to-noise ratio and reception statistics (error rate).

230140-01



WEB LINKS

[1] Norbert Schmidt, "LoRa, a Concise Introduction," Elektor 7/2016: https://elektormagazine.com/magazine/elektor-201607/29115
[2] Mathias Claussen, "LoRa GPS Tracker," Elektor 11/2020: https://elektormagazine.com/magazine/elektor-159/59110
[3] Semtech: https://semtech.com

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
- Security and encryption
- Long range/wordwide coverage
- Multi-band support

#GOCELLULAR

Adjustable Current Sink with Integrated Clock Generator

Test Power Supplies, Voltage Converters and Batteries

By Roland Stiglmayr (Germany)

A current sink, also called an electronic load, doesn't usually come standard in an electronics lab, even though it offers many benefits for the thorough testing of all sorts of power sources. Unfortunately, commercial devices are fairly expensive, so it's worthwhile taking your soldering iron and side cutters in hand and building your own electronic load.

Elektor-Labs.com BEST OF Elektor-Labs.com

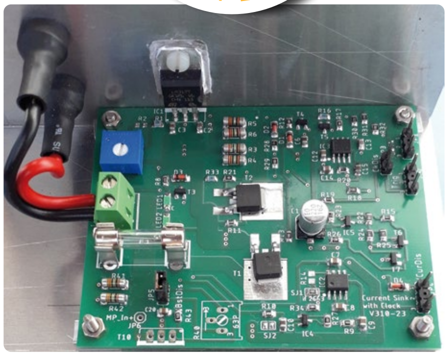
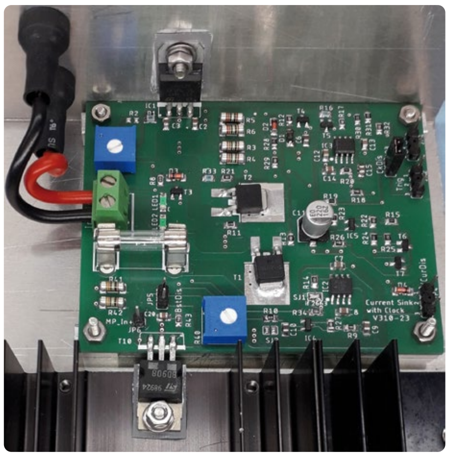



Figure 1: The current sink without the booster and with a large heatsink plate (left). The plastic screw for the LM317 was not a good idea. The fully populated board with the booster is shown on the right.

How do you test your power supplies, DC/DC converters, or batteries? You probably have a collection of power resistors that you can use to apply a load to the device under test, and you measure the current and voltage. Then you perform a series of measurements to determine the static characteristics of the control loop at different loads and input voltages. In addition, you can calculate the internal resistance of the power source from these measurements. This procedure works, but it is complicated and time-consuming.

It would be a lot better and easier to use an adjustable current sink that draws a constant current regardless of the applied voltage. However, even with an adjustable current sink, it's not possible to capture the dynamic behavior of a power supply, i.e. how it responds to fast load changes. This is actually essential for a complete assessment of a power supply. Power supplies often respond to sudden load changes with strong overshoots in the output voltage, which can cause malfunctions or even damage in the equipment connected to them. Furthermore, power supply control loops are often instable and tend to generate high-frequency oscillations, particularly with fast load changes. They have the same negative effects as overshoots.

This means that a practical current sink must be able to generate fast load changes in addition to providing a static load. This is exactly the task performed by the current sink described here. Two versions of this current sink are shown in **Figure 1**. The circuit

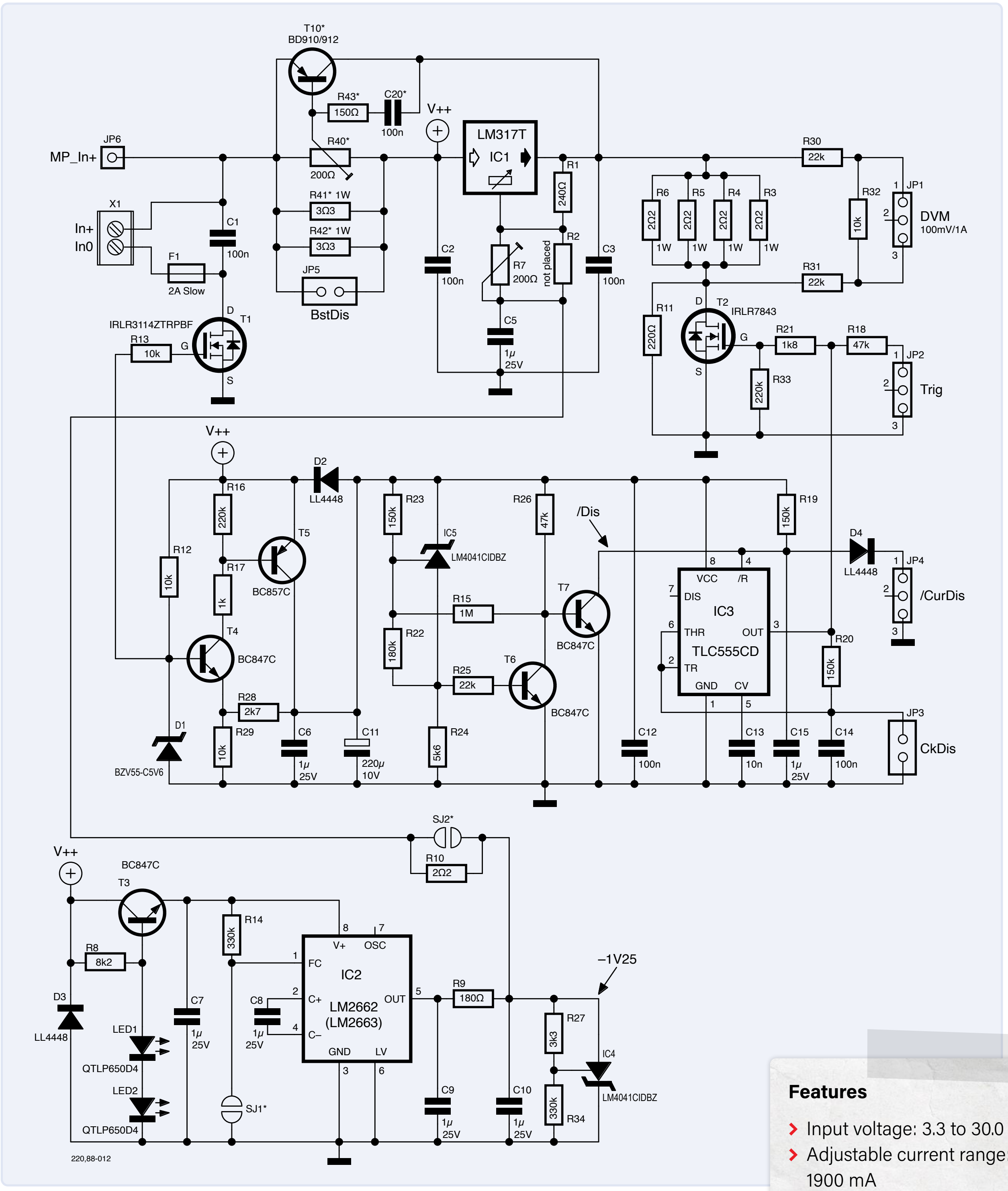


Figure 2: Schematic diagram of the current sink.

design is simple and does not contain any exotic components, and it avoids the need for a separate power supply. Depending on the heat sinks for the semiconductor devices, the current sink can handle a power dissipation of up to 18 W (or 50 W with the booster circuit) and is suitable for a maximum input voltage of 30 V. Although the maximum power dissipation may not seem so high, the ability to clock the current sink also allows a good estimation of the dynamic characteristics of relatively heavy-duty power supplies.

The specifications in the **Features** box testify to the capabilities of this current sink.

How It Works

Almost every current source or current sink is essentially based on a linear circuit that generates a constant voltage. This voltage is loaded with a fixed resistor. In accordance with Ohm's law, a constant voltage over a fixed resistance results in a constant current through the resistance. If we ignore the internal quiescent current, this current flows through the circuit

Features

- Input voltage: 3.3 to 30.0 V
- Adjustable current range: 30 to 1900 mA
- Maximum power dissipation: 18 W (50 W with booster)
- Clock (switchable): approx. 50 Hz, 50% duty cycle
- Rise time/fall time: <3 µs
- Reverse polarity protection at input
- Overtemperature protection
- Undervoltage shutdown
- Trigger signal for oscilloscope
- Output for current measurement
- Powered from the device under test

and, as a result, acts as a load on the power supply under test. This remains true even when the voltage at the input of the circuit changes, so we effectively have a load that draws a constant current under all conditions. If the voltage on the load resistance is adjustable, the current drawn by the current sink can also be adjusted. An obvious solution is to use an integrated voltage regulator to implement this circuit. Along with low complexity, this has the advantage that many voltage regulators have built-in protection functions.

The lowest output voltage possible with these voltage regulators is always the bandgap reference voltage, which is usually 1.25 V. This means that additional means are necessary to allow the current to be set to zero. However, if the regulator's feedback terminal (the reference point used by the voltage regulator to set the output voltage) is tied to -1.25 V, the output voltage with reference to circuit ground can be adjusted all the way down to 0 V. This means the current adjustment range also extends to zero. The required negative voltage can be implemented relatively easily using a charge pump followed by a voltage stabilization stage.

The easiest way to clock the load current is to use a MOSFET that switches the load resistor at the output of the voltage regulator. The clock frequency should be chosen to be close to the line (mains) frequency. This way, the power supply under test is loaded over a full half wave of the line voltage, which allows correct dimensioning of the power supply filter capacitors to be checked.

The Circuit

The circuit of the adjustable current sink is shown in **Figure 2**. The components marked with an asterisk (*) are only required for the booster circuit. The tried-and-true LM317 is used as the voltage regulator, IC1. This voltage regulator has optimal thermal properties and a maximum power dissipation of 20 W at reasonable ambient temperatures, a maximum input voltage of 37 V, and overtemperature and overcurrent protection.

IC2 is a type LM2662 voltage inverter based on a charge pump. It generates a negative voltage that is stabilized at -1.25 V by IC4 — a type LM4041 precision shunt regulator. The negative voltage at the Adjust terminal of voltage regulator IC1 is applied through trimpot R7 to set the adjustable reference point. Resistors R1 and R7 form a voltage divider, with a constant voltage of +1.25 V over R1 and a voltage of -1.25 V at the bottom end of R7. This way, the reference point of IC1 can be adjusted over the range of -1.25 V to -0.2 V. As a result, the voltage regulator's output voltage ranges from 0 V to approximately 1 V. This voltage is applied to the parallel group of load resistors, R3–R6, which determine the amount of current drawn by the circuit. The switch transistor, T2, which handles clocking of the current, is connected in series with the load resistors.

To keep the power dissipation of shunt regulator IC4 low, charge pump IC2 is operated at only 3 V. This voltage is generated by emitter-follower T3. The reference voltage is provided by two green LEDs connected in series, which have a significantly more pronounced knee voltage than a Zener diode with the same Zener voltage.

The rest of the circuit is powered by a simple voltage regulator circuit consisting of T4, T5, and Zener diode D1, which provides the reference voltage. This circuit has the advantage that it has a low voltage drop (LDO) and can thus provide the required gate voltage for T1 and T2 even with a low input voltage. The drive signal for transistor T2 comes from another good friend, a classical 555 timer (IC3). As long as capacitor C14, which is responsible for timer oscillation, is shorted out by fitting jumper JP3 (ClkDis), the output of IC3 is high and T2 is constantly switched on. This corresponds to the static operating mode. If the jumper is removed, the current is clocked. On power-up, if the input voltage is less than the minimum operating voltage, IC3 is reset via its Reset input. In this situation, the timer output is low, and no load current can flow. The timing element, R19/C15, delays the output-enable. The current sink can be disabled by applying an external low signal to JP4 (/CurDis). Among other things, this allows a time-driven discharge of a rechargeable battery to be implemented using an Arduino board.

The undervoltage shutdown is implemented with the shunt regulator, IC5 (also an LM4041), which is configured so that no current flows through it at a voltage less than 3 V, causing the timer to be persistently reset via T6 and T7. When the supply voltage rises above approximately 3.1 V, IC5 starts conducting, causing T6 to switch on and via T7 withdraw the reset of IC3.

A voltage signal proportional to the load current is available at JP1 for measurement with a voltmeter. The proportionality factor is 0.1 V/A. A signal with a peak value of around 6 V is available at JP2. In clocked mode, it can be used to trigger an oscilloscope.

The circuit input is protected against reverse-polarity connection by the MOSFET, T1. When the input voltage polarity is correct, a current flows through the body diode of T1 to allow the internal power supply to start up. As a result, T1 is fully switched on. In the event of a short circuit in the current sink, the slow fuse, F1, will blow to prevent a fire.

Higher Power

If you want to increase the power capacity of the current sink, you can connect a booster circuit ahead of the voltage regulator. This will increase the permissible power dissipation to as much as 50 W, depending on the heatsink. However, the maximum current is still the same. Unfortunately, the LM317's protection functions do not apply to the booster, so it's important to provide an adequately dimensioned heatsink. The booster circuit uses around 1.2 V of the applied voltage, so the minimum input voltage is increased to 5 V. For this reason, the booster circuit should always be disabled for voltages under 8 V by fitting a jumper on JP5 (BstDis).

The booster circuit's working principle is fairly straightforward. Part of the input current flows through the series resistors, R41 and R42. Trimpot R40 taps off a portion of the voltage drop over the series resistors and applies it to the base-emitter junction of PNP transistor T10. R40 is set so that T10 starts conducting at a current of approximately 0.6 A. As a result, T10 supplies any current above 0.6 A directly to load resistors R3–R6, while the

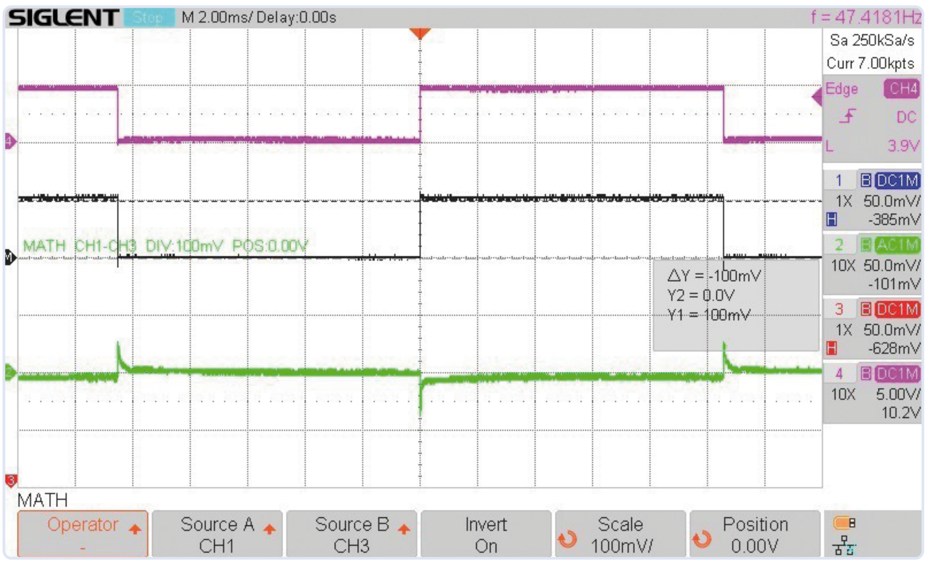


Figure 3: Dynamic behavior of a linear lab power supply at 12 V / 1 A.

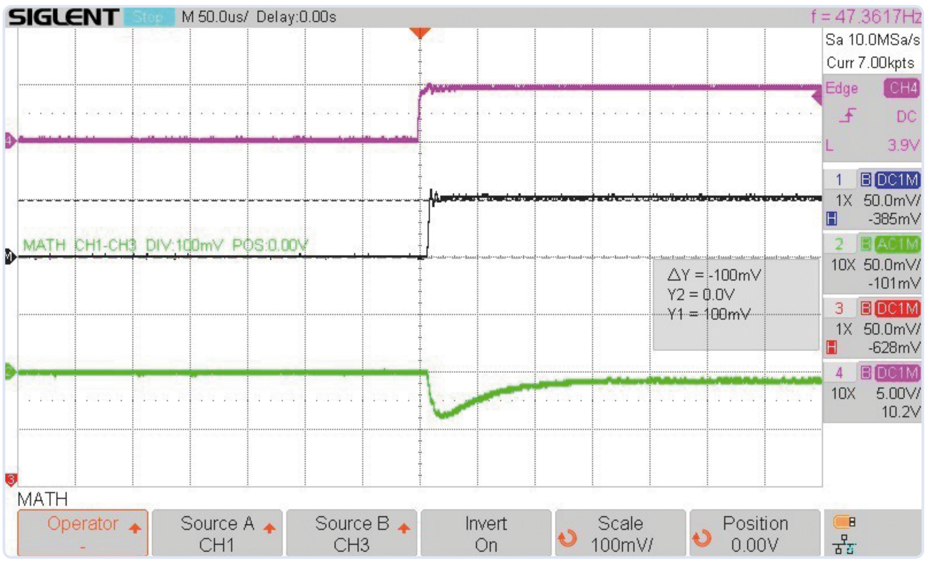


Figure 4: Step response of the lab power supply from Figure 3 to switching on the load.

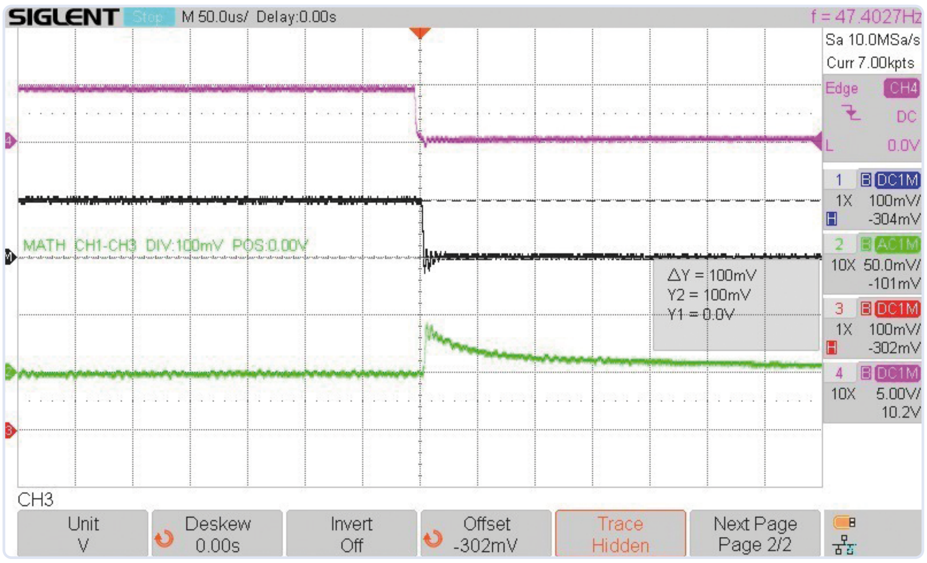


Figure 5: Step response of the lab power supply from Figure 3 to switching off the load.

LM317 continues to regulate the voltage as usual. RC network R43/C20 prevents the circuit from oscillating.

Test Procedure and Interpretation of the Results

To measure the static parameters of a power source, disable clocking by connecting a jumper at JP3. Measure the output voltage with a voltmeter connected to the output of the device under test. Measuring directly at the current sink would lead to an incorrect result due to the voltage drop over the connecting wires. Also, connect an oscilloscope configured for AC input in parallel with the voltmeter. Then, increase the current in steps while measuring the source voltage. This gives you a series of measurements that show the control deviation of the device under test. In addition, the oscilloscope gives you an indication of any oscillation tendency from the device under test. When increasing the current, you should always keep an eye on the current sink's maximum power dissipation. Up to 18 W is possible with a good heatsink for the LM317, but this is quickly reached at relatively high input voltages. Fortunately, the LM317's internal protection functions (safe operating area, SOA) protect against damage.

The next step is to measure the dynamic characteristics of the device under test using the oscilloscope. For this, clocking of the current sink must be activated. It's especially important to connect the oscilloscope directly to the power source, as otherwise the results will be falsified by voltage spikes caused by the inductance of the connecting wires. The basic measurement procedure is the same as for static measurements. The power dissipation is cut in half by the 50% duty cycle clocking, so you can use higher voltages and/or currents now. Each of the following oscillograms shows the trigger signal at the top (CH4, violet trace), the current in the middle (black trace, measured differentially at JP1 with 0.1 V/A), and the output voltage at the bottom (CH2, green trace). Also note the different horizontal and vertical resolutions in the oscillograms.

Figure 3 shows the dynamic behavior of a linear lab power supply; **Figure 4** and **Figure 5**

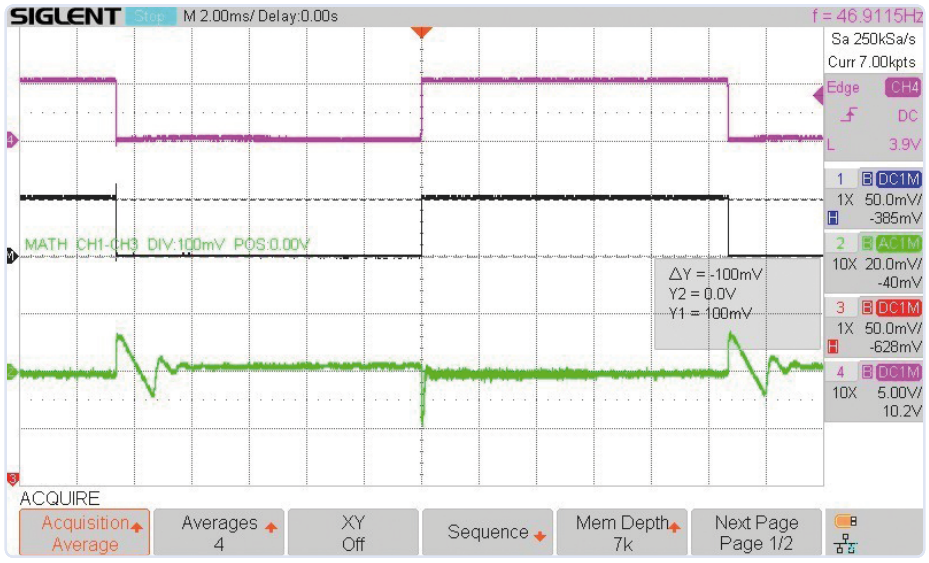


Figure 6: Dynamic behavior of a switch-mode power supply at 12 V / 1 A.

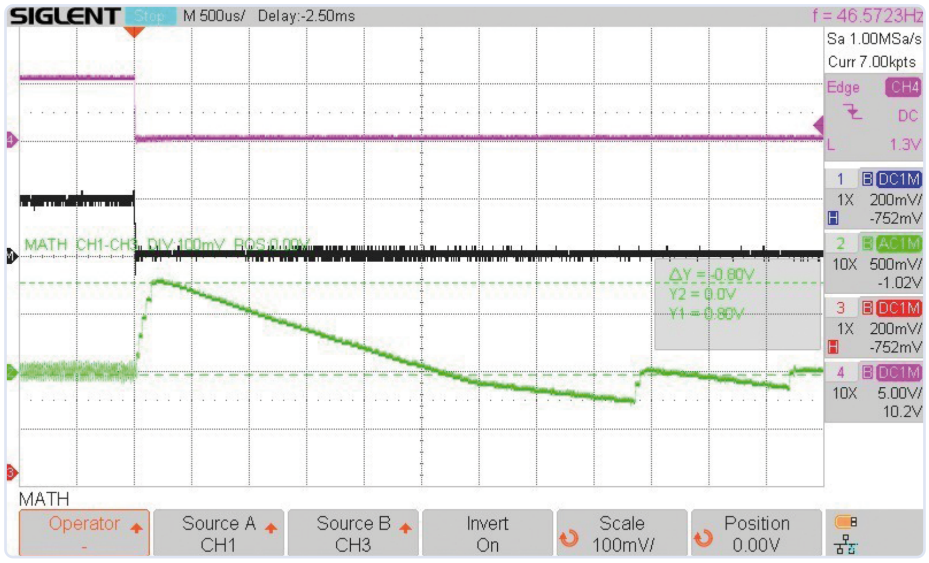


Figure 7: Step response of a flyback voltage converter to load shedding at 13 V / 1 A.

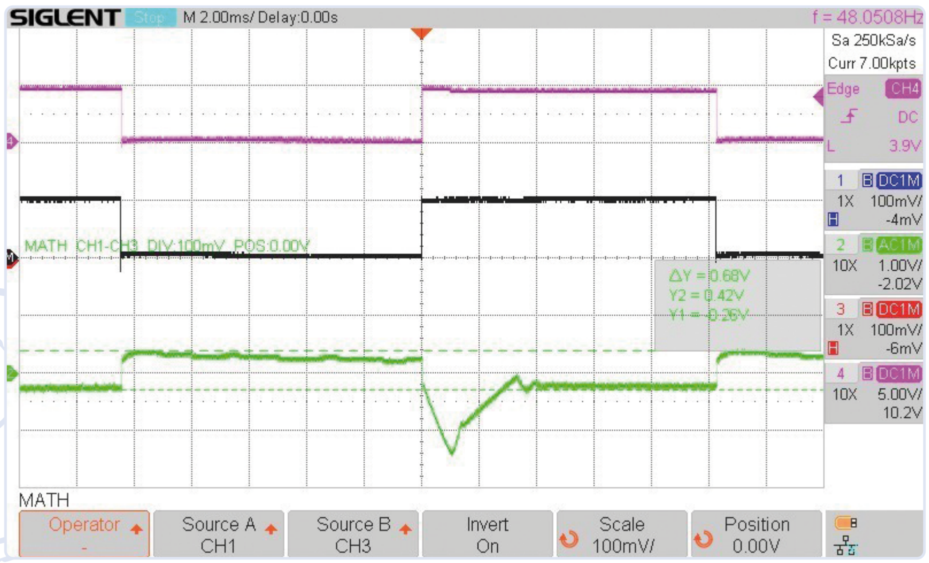


Figure 8: Dynamic behavior of a USB charger at 1 A.

show the response to step changes when the load is switched on and off, respectively. The amplitude and duration of the overshoots are the main items of interest here. **Figure 6** is the counterpart to Figure 3, but measured at a switching lab power supply. **Figure 7** shows the step response of a flyback converter to load shedding. **Figure 8** and **Figure 9** show the dynamic behavior of two USB chargers; the key item of interest here is the persistent offset of the output voltage. The control deviation, also known as the voltage drop under load, and the associated current correspond to the internal resistance of the power source.

You can also see whether the internal filtering and output damping are correctly dimensioned. If the power source has a tendency toward high-frequency oscillation, it will be reliably revealed by this measurement. If you design a power supply yourself, this current sink is an invaluable tool for optimizing the control loop's timing parameters and stability.

A dynamic measurement from a rechargeable battery provides information on its internal resistance, which is a good indication of its state of health. For this, the measurement current should not exceed a factor of 1 or 2 of the nominal capacity. The internal resistance of a cell from a lithium-ion battery with a 1500 mAh capacity is usually less than 30 mΩ, and, with larger capacities, it is usually even less (in the single-figure range). It's a good idea to measure the internal resistance of a new battery so that you can use this value later on for comparison purposes. In the oscillogram in **Figure 10,** you can see that, with an internal resistance of 5.9 mΩ per cell (R_i = 35.2 mV/(2 A × 3)), the battery is in healthy condition. Looking at **Figure 11**, on the other hand, you can see that, with an internal resistance of 53 mΩ per cell (R_i = 316 mV/(2 A × 3)), the battery is rapidly approaching the end of its useful service life.

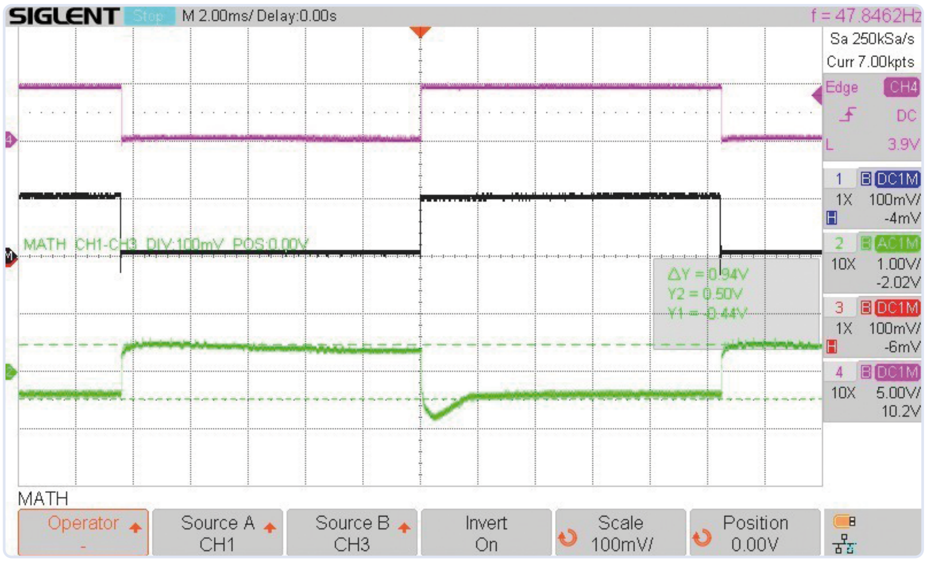


Figure 9: Dynamic behavior of another USB charger at 1 A.

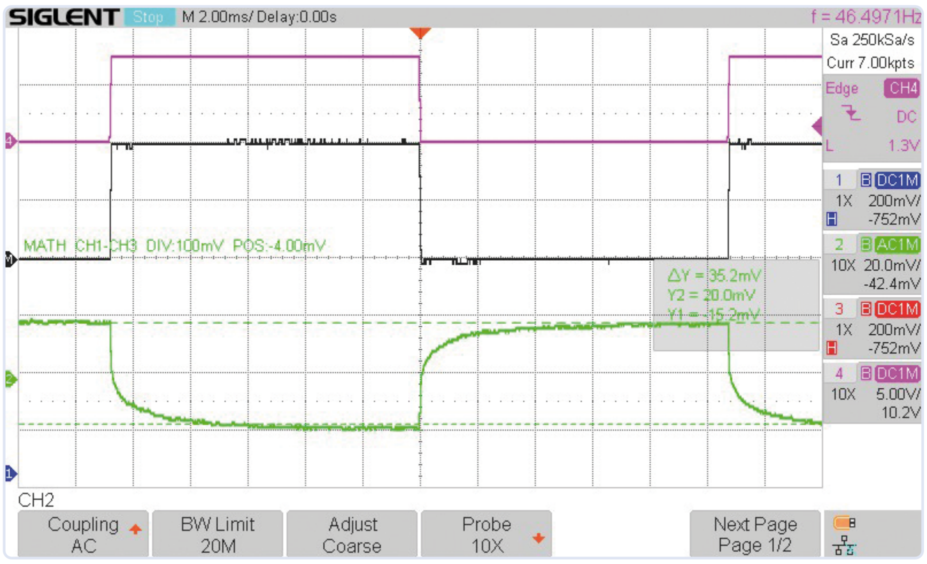


Figure 10: Measuring the internal resistance of a healthy three-cell LiPo battery at 2 A.

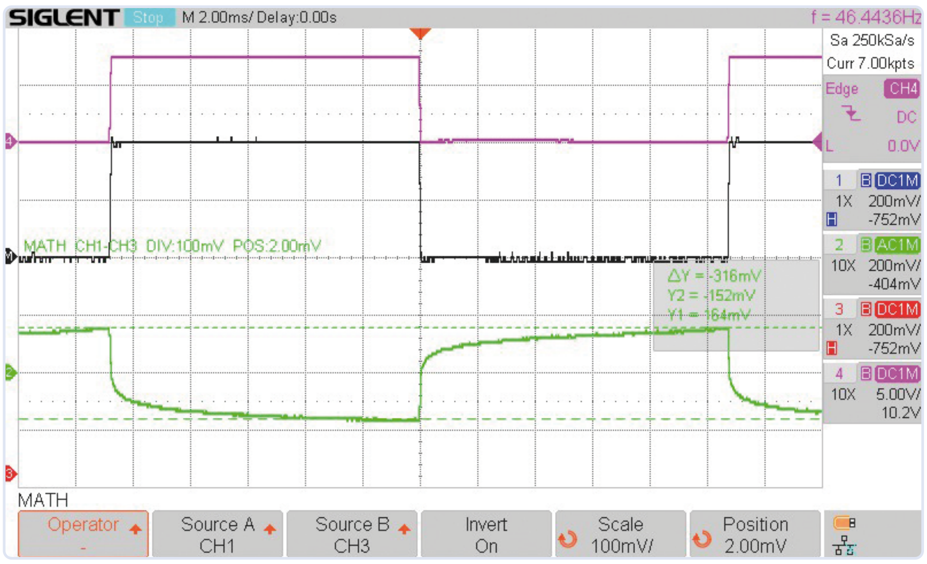


Figure 11: Measuring the internal resistance of a used three-cell LiPo battery at 2 A.

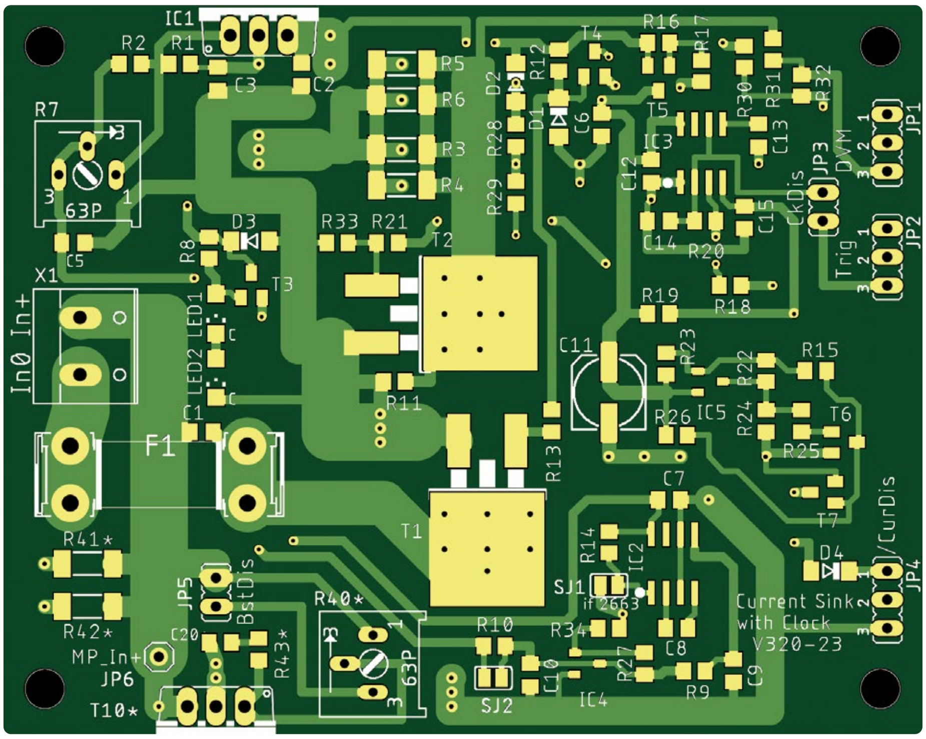


Figure 12: Board layout for the adjustable current sink with booster.

Construction

A PCB layout has been designed that also supports the addition of the booster circuit. The layout in **Figure 12**, as well as the schematic diagram, can be downloaded as a PDF from [1]. The Eagle files are also included in the download.

The footprints for T1 and T2 allow placement of both DPAK and D2PAK components. For IC2, you can use either an LM2662 or an LM2663, so there shouldn't be any problems procuring the components. If you don't need the booster, you can omit the components marked with an asterisk (*). In any case, you should pay careful attention to the heatsinks for IC1 and T10, because the reliability of the current sink essentially depends on them. The heatsinks' thermal resistance must be less than 2 K/W, and the semiconductor devices must be mounted using high-quality thermal pads and metallic screws with suitable washers, with proper attention to insulation.

Getting Started with the Current Sink

If you use an LM2663 for IC2, solder bridge SJ1 must be closed. Fit a jumper on JP5 (BstDis). Using trimpot R7, adjust the current sink to minimum current while connected to an adjustable power supply with the output voltage set to 3 V and current-limiting set to 100 mA. The current sink should draw a quiescent current of approximately 10 mA. Next, increase the voltage to 3.3 V. Using trimpot R7, set the current to around 80 mA. Then, reduce the voltage again. At around 3.1 V, the current should drop back to the quiescent level. This indicates that the undervoltage shutdown is working properly. With an input voltage of 5 V and trimpot R7 set to its minimum value, the current should be less than 30 mA. Otherwise, the current can be reduced by closing solder bridge SJ2.

To put the booster into operation, turn trimpot R40 counter-clockwise to prevent any base current flowing into T10, then remove the jumper on JP5. With an input voltage of 15 V and a current of 1 A, adjust trimpot R40 so that the voltage regulator's input current is 0.6 A. You can measure the voltage regulator's input current from the voltage drop over R41 or R42 at JP5. There, 1.0 V corresponds to 0.6 A. Next, increase the input voltage to 25 V and the current to 2 A. If necessary, you can adjust the setting after the current sink has warmed up.

RG, Translated by Kenneth Cox — 220388-01

About the Author

Roland Stiglmayr studied computer science in the 1970s and has 40 years' experience in R&D. His main focuses were developing computer mainframes, fiber-optic data transmission systems, remote radio heads for wireless networks, and contactless energy transmission systems. He is currently active as a consultant, with a particular passion for knowledge transfer.

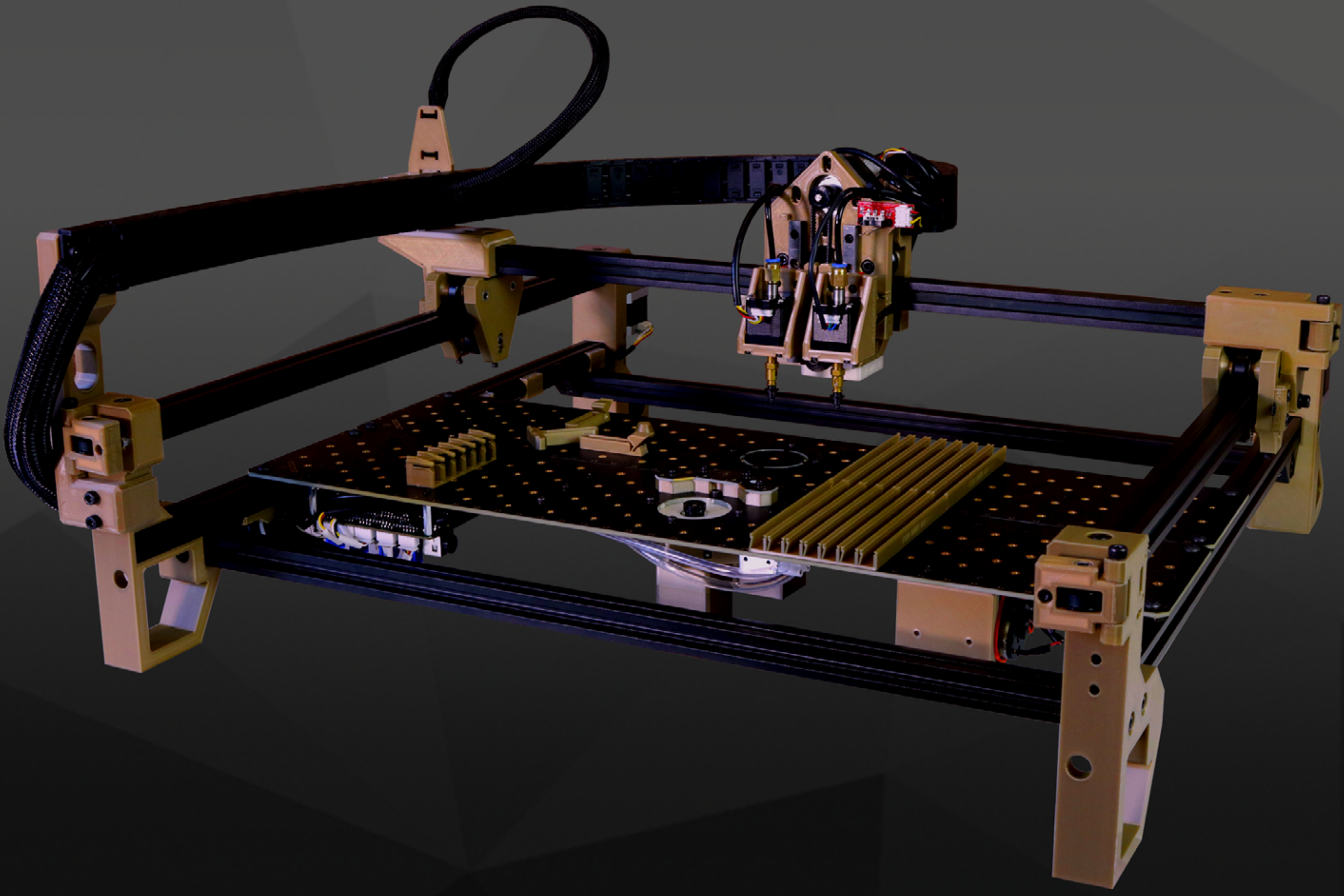
Questions or Comments?

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

WEB LINK

[1] Project downloads at Elektor Labs: www.elektormagazine.com/labs/adjustable-current-sink-with-integrated-clock-generator

VS



It’s no contest.

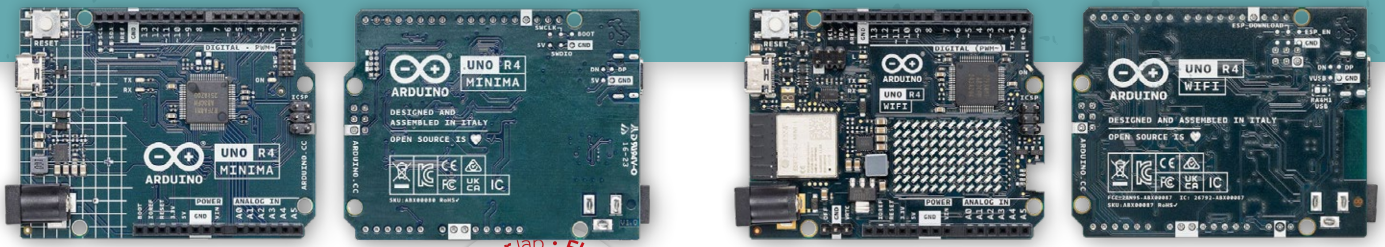
The LumenPnP Desktop Pick & Place Machine assembles your boards, so you never need to use tweezers again

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

Opulo™

Opulo.io

Two New Arduino UNO R4 Boards: Minima and WiFi



By Clemens Valens (Elektor)

The powerful Arduino UNO R4 is the newest member of the iconic Arduino UNO family. It even comes in two versions. Let's take a look at the Minima and WiFi.

Announced a few months ago [1], the Arduino UNO R4 Minima [2] and the Arduino UNO R4 WiFi [3] have now been released officially. So, now that they really exist, let's take a closer look at them.

The Renesas Family

A few months ago, Arduino released the Portenta C33 board, which has an ARM Cortex-M33 microcontroller from Renesas: the RA6M5. The two new boards also feature a device from Renesas, the RA4M1 (**Figure 1**). This 32-bit ARM Cortex-M4 runs at 48 MHz and has 32 KB of RAM and 256 KB of flash memory. It seems that a family of Renesas boards is starting to develop.

Interesting to note is that the RA4M1 can work with a power supply of up to 5 V, whereas most other ARM-type microcontrollers require 3.3 V. This makes the MCU a suitable candidate for enhancing the 5 V, 8-bit AVR-based family, of which the Arduino UNO R3 is a famous member.

More Peripherals on the UNO R4

Replacing an old 8-bit, 28-pin controller with a modern 32-bit 64-pin device has, as you may expect, an impact on the complexity of the product. However, this touches mainly the software part, as indicated by the MCU's ±1,400-page-long datasheet (versus fewer than 300 for the ATmega328). The UNO R4 Minima board is of similar design complexity as the R3. The UNO R4 WiFi is a denser board, as it uses the Minima's empty board space for an 8×12 LED matrix and an ESP32-S3-MINI-1 module.

The datasheet is so long because the Renesas MCU features many more peripherals than the Microchip ATmega328. Not all of these are supported by the R4s because not all the pins of the MCU are

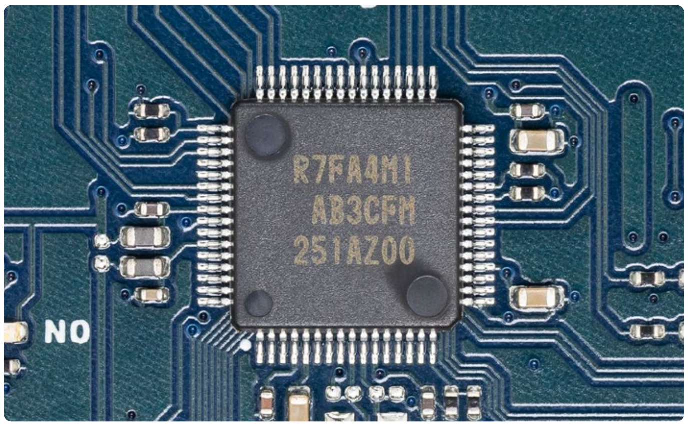


Figure 1: The UNO R4 boards are powered by a Renesas R(7F)A4M1 microcontroller.

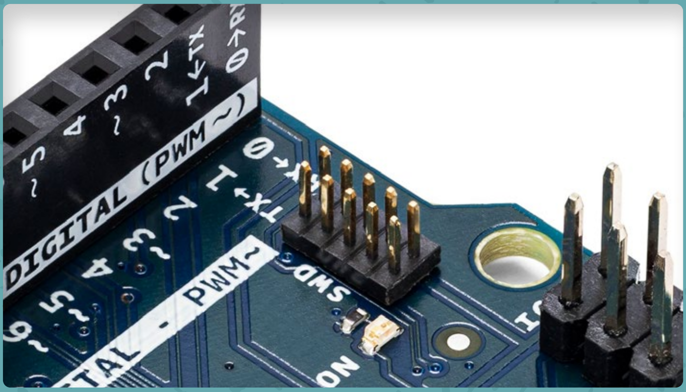


Figure 2: The SWD connector on the UNO R4 Minima.

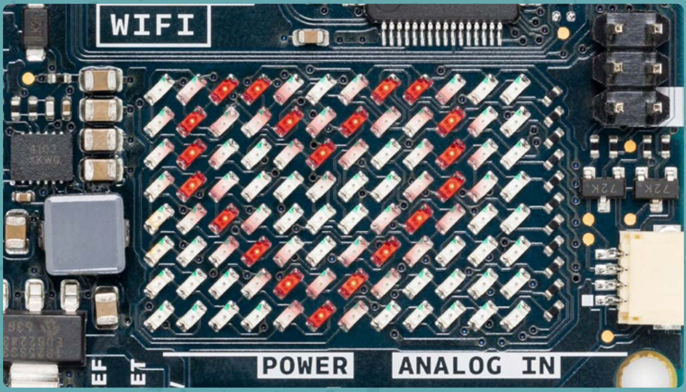


Figure 3: The Arduino UNO R4 WiFi has an 8×12 matrix of red LEDs.

accessible, but a few nice ones are. These include a CAN bus and USB 2.0 Full-Speed (host or device).

A USB-C connector replaces the USB-B connector.

Serial and Serial1

The RA4M1 integrates two SPI ports, two I²C ports, and four serial communication interfaces (SCIs). An SCI can be either an UART, an I²C master, or a simple SPI port (i.e. up to six I²C or SPI ports), and even a smartcard interface. According to Arduino, these ports are available, at least to some extent . However, looking at the Minima's schematic, only one I²C port appears to be connected. Pins A4, A5, D4, and D5 expose a second SPI port, even though the board specifications mention only one.

Important to note is that the traditional `Serial` port on both USB and Pins 0 and 1 has been split into `Serial` and `Serial1`. By default, `Serial` refers to the UNO R4's USB port, while Pins 0 and 1 are for `Serial1`. It is possible to map `Serial` to Pins 0 and 1 by defining `NO_USB` before *explicitely* including *Arduino.h*, but this disconnects the serial port from the USB port. Programs and libraries that rely on the UNO R3 way may encounter problems on the UNO R4.

The UNO R4 Has Better Analog

The UNO R4 features a 12-bit digital-to-analog converter (DAC) for producing real analog signals instead of PWM-based surrogates. There's also an opamp and a comparator together with an internal 8-bit DAC, and the analog-to-digital converter is 14 bits wide instead of the 10 on the UNO R3. The *analogWave* library was added to make using the DAC easy. Generating a sine, sawtooth, or square wave is as easy as calling a library function. Of course, you can do much more with it. Therefore, on the analog side, the UNO R4 has much more to offer than the UNO R3.

Added Software Complexity

Regarding the Arduino IDE, switching to a new, hitherto unsupported processor family also implies adding software support. As we have learned over the years in our computer-controlled world, new software tends to introduce issues, so it will probably take some time before the UNO R4 experience will become as smooth as that of the UNO R3.

Luckily, the UNO R4 Minima makes resolving issues somewhat easier, as it has an SWD interface, which allows for serious (serial) debugging (**Figure 2**). The UNO R4 WiFi even takes this a step further as its ESP32-S3 module can act as an on-board CMSIS-DAP debugger.

Wi-Fi Modem

Now that we mentioned the Arduino UNO R4 WiFi, let's see in what way it differs from the UNO R4 Minima. Firstly, there is, of course, the Wi-Fi & Bluetooth LE module — an ESP32-S3 from Espressif. It communicates with the MCU over a serial port (`Serial2`) in AT-command mode. The Wi-Fi module's other pins are exposed as tiny solder pads. Reprogramming the module is possible, as the required pins for this are accessible on a 2×3 header (and on the bottom side of the PCB). The new Arduino library *WiFiS3* provides high-level software support for the module.

LED Matrix

A 96-pixel (8×12) red LED matrix (**Figure 3**) enables users to plot data, create animations and provide more complex and sophisticated feedback in projects. A new library provides functions to show animations on it (before the LED matrix library became available, I had already written a little program to display a scrolling message on the UNO R4 WiFi - you can download it from GitHub [4]). A web tool for designing animations has also been announced.

The matrix uses Charlieplexing [5] to connect the 96 LEDs to only 11 GPIO ports (D28 to D38 in Arduino notation). This means that only a few LEDs can be active at any one time, as pixels consist of two LEDs connected antiparallel, and pixels share ports. ‚Because the human eye is slow, fast time-multiplexing tricks the brain into seeing complete images. The refresh rate must be high for this when serving almost one hundred pixels. The timer-driven library therefore runs at 10 kHz.

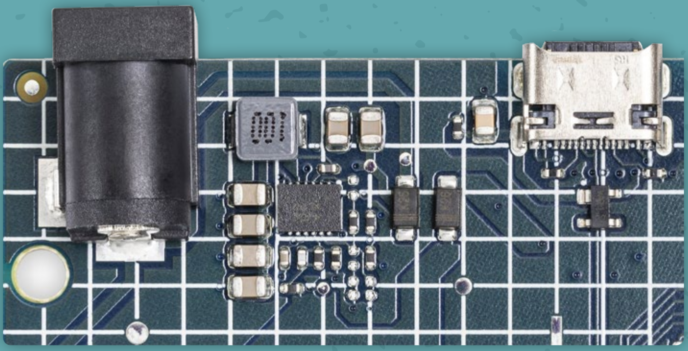


Figure 4: The boards' input voltage range is 6 V to 24 V.

Video: Learn about the Arduino UNO R4 from Arduino and Renesas engineers https://youtu.be/vrZ__aJnFd-Q

A Second I²C Port

The UNO R4 WiFi features two I²C ports (`Wire` & `Wire1`). A Qwiic-compatible (SparkFun standard) I²C connector provides access to the second port. These connectorsmay be annoyingly tiny, but, as the Qwiic standard imposes a 3.3 V supply voltage, the board comes with a level shifter. This means that the board can handle both 5 V and 3.3 V I²C communication.

Power Supply

When not powered by its USB port, the power supply on the Arduino UNO R3 is a basic linear voltage regulator. On the UNO R4 boards, this has been replaced by a switching regulator. It allows a much wider input voltage range of from 6 V to 24 V (**Figure 4**). The regulator, also a Renesas product, can deliver up to 1.2 A with an efficiency of around 90%.

Note that the UNO R4 WiFi also has a connector to power the real-time clock (RTC).

Conclusion

The two Arduino UNO R4 boards, the Minima and the WiFi, look like credible successors to the UNO R3. They have the same outline, extension connectors, and full 5-volt I/O. The UNO R4 WiFi is a bit like a UNO R4 Minima with an extension shield built in. With its Wi-Fi module, LED matrix and Qwiic connector, it's easy to build IoT applications.

The UNO R4 is way more powerful than the UNO R3, with more of everything, from memory to peripherals to operating speed. I had to manually install a special Arduino UNO R4 boards package in the IDE 1.8.19 before I could do any programming. The IDE v2 detects the UNO R4 and proposes to install the required software. Therefore, to most users, the UNO R4 will be a drop-in replacement for the UNO R3.

The UNO R4 is a Great Step Forward

Finally, very interesting is the debugging connector on the Minima and the CMSIS-DAP debug capabilities of the ESP32-S3 module on the WiFi, a feature many developers have been waiting for since about the beginning of Arduino. This takes the Arduino UNO into the pro development arena, finally. Therefore, the Arduino UNO R4 can be considered a powerful step forward.

230443-01

Questions or Comments?

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

Related Products

- **Arduino UNO R4 Minima**
  https://elektor.com/20527
- **Arduino UNO R4 WiFi**
  https://elektor.com/20528

WEB LINKS

[1] Clemens Valens, "The Arduino UNO R4 is Coming," elektormagazine.com: https://elektormagazine.com/news/arduino-uno-r4
[2] Arduino UNO R4 Minima: https://elektor.com/arduino-uno-r4-minima
[3] Arduino UNO R4 WiFi: https://elektor.com/arduino-uno-r4-wifi
[4] GitHub repository: https://github.com/ClemensAtElektor
[5] Charlieplexing: https://en.wikipedia.org/wiki/Charlieplexing

Logarithmic Potentiometers

They're Exponential!

By Joseph Kreutz (Germany)

Ideally, when you turn a volume control on an amplifier or a mixer desk, you want there to be a linear relationship between the angle of the control and the amount of sensation that your ear perceives. A result that is close to this ideal is obtained thanks to "logarithmic" potentiometers, which are actually exponential. Let's look at this a bit more closely.

Nature designed us to perceive acoustic signals whose range of amplitudes varies from 1 part to 1,000,000 parts and more, which is represented by a 120 dB ratio or more. The so-called Weber-Fechner law specifies

sensation = k • log(physical stimulation)

Here, sensation is the impression that we perceive, and physical stimulation is the acoustic pressure at our ears [1]. The k factor is specific to each one of us because our hearing characteristics are just as unique as your hair or eye color. Your hearing sensitivity depends on the frequency, the effective amplitude of the audio stimulation, and on your age, because hearing degrades with age.

The lower limit of human hearing has been measured at a sound pressure level of 20 µPa, which has been fixed as the 0 dB value on the scale of acoustic pressures, expressed in decibels. It is also useful to know that an acoustic level approaching 130 dB begins to cause physical pain. Regular exposure to sound levels above 100 dB can cause permanent hearing damage.

The Ideal Volume Potentiometer

In order for the sensation to be as proportional as possible to the position (rotation or translation) of the control device, the law respected by a potentiometer should be complementary to a logarithmic function. An exponential function suits this requirement best. We can define a parameter to represent the position of a potentiometer:

$$\alpha = \frac{\text{Effective Rotation}}{\text{Maximum Rotation}} = \frac{\text{Effective Translation}}{\text{Maximum Translation}} \quad \text{(I)}$$

As this control has a minimum (in principle, 0) and a maximum (in principle, 1), if follows that $0 \leq \alpha \leq 1$. The ideal potentiometer should thus have a characteristic corresponding to

$$S = S_{ref} \bullet 10^{\alpha \bullet F_p} = 20 \bullet 10^{-6} \bullet 10^{5\alpha}$$

S_{ref} is the lower limit of the sensitivity of the human ear, thus 20 µPa, and F_p is a parameter calculated for a maximum volume at around 100 dB above the lower threshold. Reference [2] recommends a range of variation of 90 dB for an audio volume control, so, a value very close to this.

Potentiometers in the Real World

The construction of an ideal potentiometer, one that's exactly exponential, might be possible, but the technical complexity would make it an expensive component. Manufacturers have overcome these difficulties by making tracks with two or three sections of different resistivity to create enough linear segments to approximate an exponential function. This compromise is deemed perfectly satisfactory for current applications.

Linear Potentiometer Loaded with a Resistance

Another way of approximating an exponential response is to load a linear potentiometer with a resistance connected between its wiper and ground, as illustrated by the circuit in **Figure 1**. The response thus obtained is characterized by the following function:

$$V_{OUT} / V_{IN} = \alpha \bullet \gamma / (\alpha \bullet (1-\alpha)+\gamma) \quad \text{(II)}$$

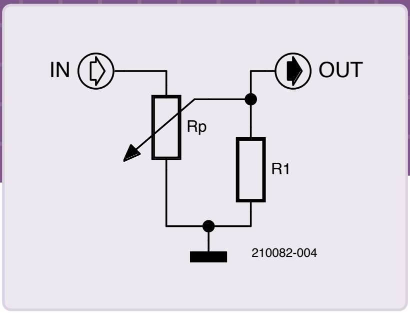


Figure 1: By adding a resistor to a linear potentiometer as shown here, it is possible to approximate an exponential response.

Here, α represents the rotation or translation of the control element defined by (II), and γ the ratio between load resistor R1 and the total resistance of potentiometer Rp. For example, if Rp = 1 MΩ and R1 = 39 kΩ, then $\gamma = R1 / Rp = 0.039$.

The graphed curves in **Figure 2** represent the ideal exponential function (green), and the function obtained with a γ value of 0.039 (red). This choice of γ and the corresponding resistance values give results closest to the exponential function. There is, however, a discrepancy between the ideal curve and this approximation, which appears at very low levels, and the attenuation shows a small but abrupt jump at the start of the curve. For this reason, this solution is not recommended for volume control in hi-fi equipment [2].

Taking into account the parameters defined by Equations I and II, load resistance $R_L(\alpha)$, as seen by the stage preceding the R1 / potentiometer assembly, will be

$$R_L(\alpha) = Rp \cdot \alpha \cdot (1 - \alpha) + \gamma / (\alpha + \gamma)$$

The values are respectively equal to Rp for α = 0 and Rp//R1 for α = 1. This equation presupposes that the impedance of the stage that follows the volume control is sufficiently high to have only a negligible effect. The stage that precedes the volume control must be capable of powering the volume control.

Also note that the total value of a potentiometer's resistance Rp is generally specified with a tolerance of ±20%, which makes it one of the components with the highest tolerances.

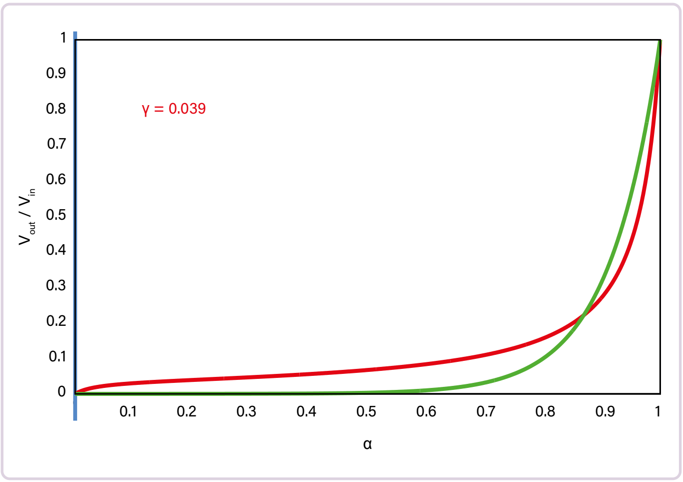


Figure 2: The ideal exponential curve (green), and the actual curve obtained with γ value of 0.039 (red). This value of γ gives the closest results to an exponential function.

Thermal Noise

An ideal resistor produces a thermal noise voltage equal to

$$e_n = \sqrt{(4 \cdot k \cdot T \cdot R \cdot B)} \quad \text{(III)}$$

where $k = 1.38 \cdot 10^{-23}$ (Boltzmann's constant), T the temperature in kelvin (here 298.15°K being 25°C), R the resistance value in ohms and B the frequency bandwidth (here 20 kHz) at which the measurement is taken.

The series resistance introduced by a potentiometer in the circuit of Figure 1 contributes at most a quarter of its total value $(\alpha \cdot (1-\alpha) \cdot Rp$ with α = 0.5), so 250 kΩ if Rp = 1 MΩ, and the corresponding noise voltage will amount to

$$e_n = \sqrt{(4 \cdot 1.38 \cdot 10^{-23} \cdot 298.15 \cdot 250{,}000 \cdot 2 \cdot 10^4)} \approx 9.07\ \mu V$$

The maximum noise voltage will be attenuated by the divider consisting of the potentiometer's series resistance and of R1 (39 kΩ). Its contribution will thus be reduced to 1.22 µV.

The noise generated by the 39 kΩ resistor must also be considered. It will be up to 3.58 µV for a temperature of 25°C and a bandwidth of 20 kHz. Its contribution to total noise will be 3.1 µV. Thus, the maximum thermal noise voltage is

$$e_{n,max} = 3.1 \cdot 10^{-6} + 1.22 \cdot 10^{-6} = 4.32\ \mu V$$

This represents an improvement of 6.4 dB over a potentiometer alone.

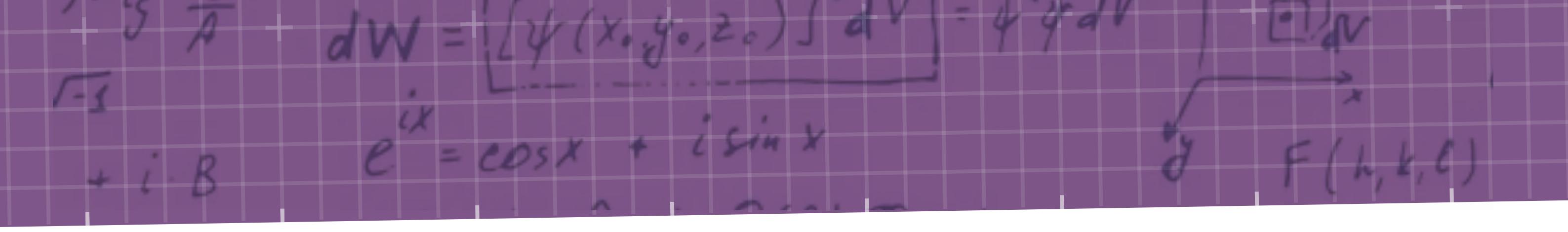
In most commercial potentiometers, the track is made of carbon, which produces a noise level higher than that calculated with Equation (III). Some manufacturers offer potentiometers with a track made of plastic or "cermet," a combination of a metal and a ceramic. They are not as sensitive in use and generally present a noise voltage lower than carbon potentiometers. Of course, it is better to choose a low-noise component for R1, for example, a metal film resistor.

A generally evident conclusion from the foregoing is that in electronics, in order to avoid thermal noise, the circuits should use resistors of the lowest possible value.

Comparison of Solutions

The difficulties of constructing a potentiometer presenting an exponential characteristic have been overcome by the two methods above. Which of them is better?

In the case of a volume control for consumer equipment, the approximation by linear segments offered by manufacturers in their so-called "logarithmic" potentiometers gives a perfectly acceptable solution



because at low levels the attenuation is reasonably close to an ideal exponential and tends towards an effectively zero value. For this, consult the manufacturers' characteristic curves.

For a volume control for use in a mixing desk or at the input of an amplifier, a linear potentiometer loaded with a resistor will give complete satisfaction. Even though the control shows some anomalies at low levels, the use of a resistor with a linear potentiometer also allows a substantial reduction in the level of thermal noise. This is a significant advantage in many applications, particularly for professional equipment.

210082-01

WEB LINKS

[1] Equal-loudness contour: https://en.wikipedia.org/wiki/Equal-loudness_contour
[2] D. Self, "Small Signal Audio Design (2nd Edition)," Routledge, 2014: https://elektor.com/18046

Motor Driver Breakout Board

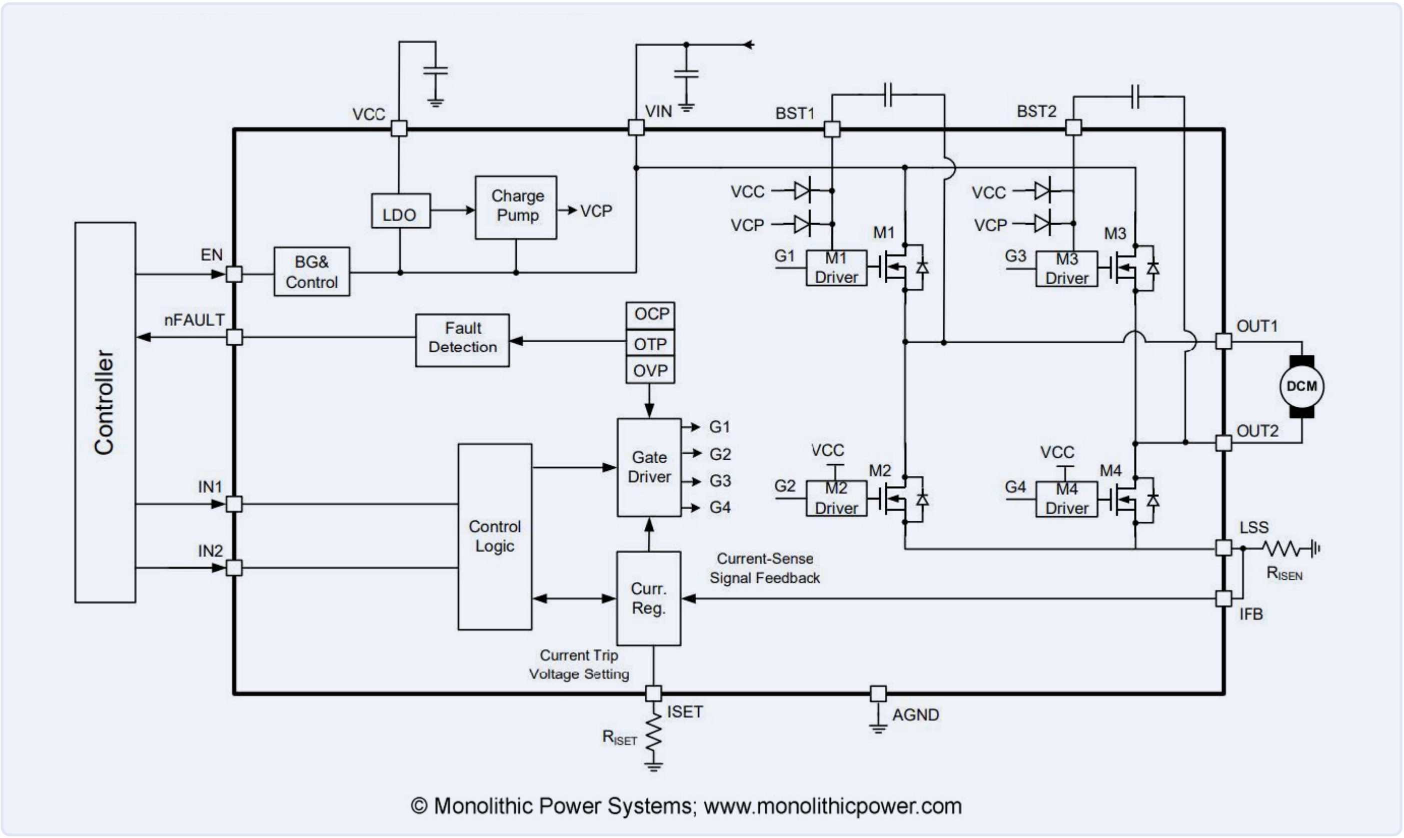
A BoB for a 5 A DC Motor Driver with a 3×3 mm Size

By Edwin van den Oetelaar (The Netherlands)

Today's makers and students are often in a hurry and lack the skills to design and solder electronic circuits. They are more into "shields," and handling a soldering iron is a challenge. So, I've developed a breakout board for the Monolithic Power Systems MP6619 motor driver, which can be used well for one's own motor-based projects.

Figure 1: Functional block diagram. (Source: Monolithic Power Systems [1])

▼



This article is a journey of discovery to a MP6619 breakout board (BoB). This chip is a new compact H-bridge for controlling low-cost brushed DC electric motors and solenoids. These DC motors are the most accessible and affordable for hobbyists involved in building small robots, modelmaking, and creative projects. This is a story about the educational journey I had to go through to develop and test this breakout board.

The Adventure Begins

One day, I received an email from a component distributor about a brand-new H-bridge component: the MP6619. **Figure 1** shows its internal block diagram. This tiny "miracle" of just 3×3 mm with impressive specifications such as 24 V and 5 A, and other aspects, caught my attention (see the **Chip Features** frame). How can this be possible? So much current, no heatsink, and all protection features in such a miniature package? This must be marketing talk, I thought. The alternatives are easily 10 times larger.

Chip Features

- It's not just marketing talk: The chip can indeed handle 24 V and 5 A. It is well protected against overload, short circuit, and reliably recovers from fault situations.
- Adding a choke coil or a filter improves performance significantly. The need for that depends on your application and the motor.
- The integrated N-channel MOSFETs have a low $R_{ds(on)}$, of 65 mΩ, which is essential for limiting dissipation. The necessary boost circuits to drive their gates are also integrated. Their capacitors are connected externally.
- The MP6619 chip is produced without bonding wires to achieve a good thermal performance in a QFN-19 package.
- The footprint's islands are extra long in order to dissipate heat from the chip to the PCB.
- The PCB must be carefully designed, taking into account not only electrical but also thermal requirements.

But, my curiosity was piqued, and I wanted to know if this was true and whether it would be useful for my projects. So, I ordered a few. After just two days, they were on my desk: Wow, they were tiny! Only 3×3 mm with 0.2 mm between the contacts and no legs. What have I gotten myself into?

I dive headfirst into this subject. Despite the low price of around €2.50, the small dimensions and unusual footprint of the MP6619 turned out not to be directly applicable to my projects. A BoB was needed that would make the MP6619 more manageable and suitable for testing and integration.

But, there was no BoB available for this tiny Quad, Flat, No-lead, 19-contact (QFN-19) footprint. So, I had to develop such a board myself. My adventure with the MP6619 H-bridge began!

Designed with KiCad

To develop an MP6619 breakout board, I started by studying the datasheet [1] and drawing the BoB schematic in KiCad. First, I drew the symbol and footprint (these were not available in the library or SnapEDA), and then the layout.

I paid extra attention to the placement of the $R_{(i\text{-sense})}$ shunt resistor and the connection of power pads. This shunt resistor is used to measure the MOSFET drivers' current. This is crucial for the proper functioning of the breakout board. Voltage drop across the traces should be considered, too, during the PCB layout process.

The OCP (Over-Current Protection) is adjustable with an external resistor. Without that, the chip switches off when the voltage across the shunt resistor reaches 200 mV. The datasheet suggests a shunt of 0.04 Ω, which limits the maximum current to 5 A ($V = I \times R$). By adding an $R_{(i\text{-set})}$ of 80 kΩ, the limit is set to 100 mV, allowing you to reduce the current limit when using the same shunt resistor. Other resistor values are also possible. I chose a shunt of 0.05 Ω for my first test because I had it in my drawer. This results in an output current limit of 4 A.

I quickly made the first version of the BoB PCB layout without in-depth study, as I wanted to start testing soon, but I rejoiced too soon. The first BoB included two MP6619 chips, LEDs, resistors, capacitors, and connectors. At that time, I still thought PCB terminal blocks would be useful. Four copper layers were needed, I placed many vias, a ground plane, and used the widest possible traces due to the high currents. That was a day's work. A week after sending the data to a Chinese manufacturer, I had the boards on my workbench. Soldering the QFN was the greatest challenge, as the chips were smaller than anything I was used to.

To get the QFN in place with all connections correct, a stereo microscope, a good temperature-controlled soldering iron, a hot-air soldering iron, plenty of flux, and even more patience were required. Once the QFN was in place and the sweat wiped from the forehead,

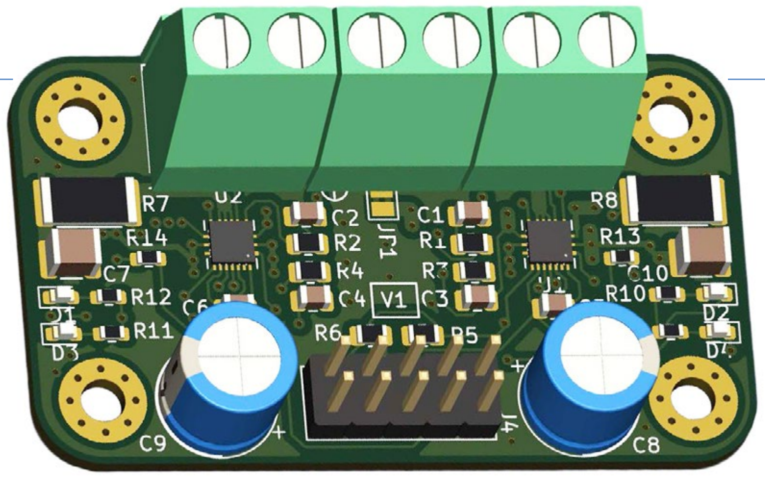


Figure 2: 3D model of the first PCB.

those other "normal" SMD components were child's play; they seemed gigantic compared to the QFN. The solder island for the PCB terminal block is larger than the entire QFN chip (**Figure 2**)! I posted this first version on the Elektor Labs website [2].

First Tests

The first tests were done with a small DC motor [3] that was on my workbench. It was a simple 12 V model with a stall current of under 5 A (**Figure 3**).

The MP6619 internally consists of two half-bridges, with each half-bridge having one input (IN1, IN2). The chip itself ensures the correct timing of the high-side and low-side MOSFETs so that no short-circuit between the power rails occurs.

The chip is protected against all typical problems. Its integrated UVLO (under voltage lockout), OVP (overvoltage protection), OCP (overcurrent protection), and OTP (over temperature protection) should safely handle nearly all error situations, which we'll see!

Connecting IN1, IN2, and EN

The inputs are equipped with pull-down resistors. As soon as the EN input receives a High level, the chip is activated, and the internal voltage regulator starts operating. At that moment, the MP6619's VCC output provides a voltage of 5 V. I used a small 100 nF decoupling capacitor to stabilize this voltage. If the 5 V is up, the N-channel MOSFETs also become active. Depending on the input levels at IN1 and IN2, either the high-side or low-side MOSFETs are switched.

It is safe to have 3.3 V to 5 V logic levels at the inputs. So it is no problem to connect the BoB to an ESP32 or an Arduino. For my initial tests, I used switches to apply 3.3 V to the inputs, and connected the motor to the BoB's outputs.

Partial Success

My motor didn't want to spin, the nFault output immediately went low (error indication LED turned on), but I didn't have a short circuit at the output. Significantly reducing the input voltage eventually allowed me to slowly rotate the motor counterclockwise and clockwise. It's important to note that I didn't provide any PWM speed control yet.

I experimented with decreasing the shunt resistance and increasing the R_{set} resistor to get the motor moving, which ultimately resulted in a "human error" — 12 V on IN1, a burnt BoB as result, and a deep sigh of frustration. The PCB went up in smoke, and I needed to stuff a new BoB with components.

Another BoB and More Tests

Eventually, I found out that the motor is equipped with noise suppression capacitors connected in parallel to the commutator. As a result, the motor behaves less inductively and more capacitively. This causes the OCP to immediately kick in when there is a current spike to the motor. This "overload" causes a timeout of 1 ms before the MP6619 tries again. This process repeats again and again, resulting in a soft, squeaking sound from the motor.

Removing these capacitors partially helps to control the motor at somewhat higher voltages. Meanwhile, after studying the datasheet more carefully, I also had improvement plans for my BoB design and decided to create a new version.

Figure 3: The first tests were done with a DC planetary geared motor. (Source: E-S Motor / RobotShop [3])

New Design and Layout

For my second version, the design and the layout had to be adjusted. I made the following changes:

- Only one MP6619 chip.
- The position of the shunt was revised.
- A 0.1" pin header instead of terminal blocks; no mounting holes.

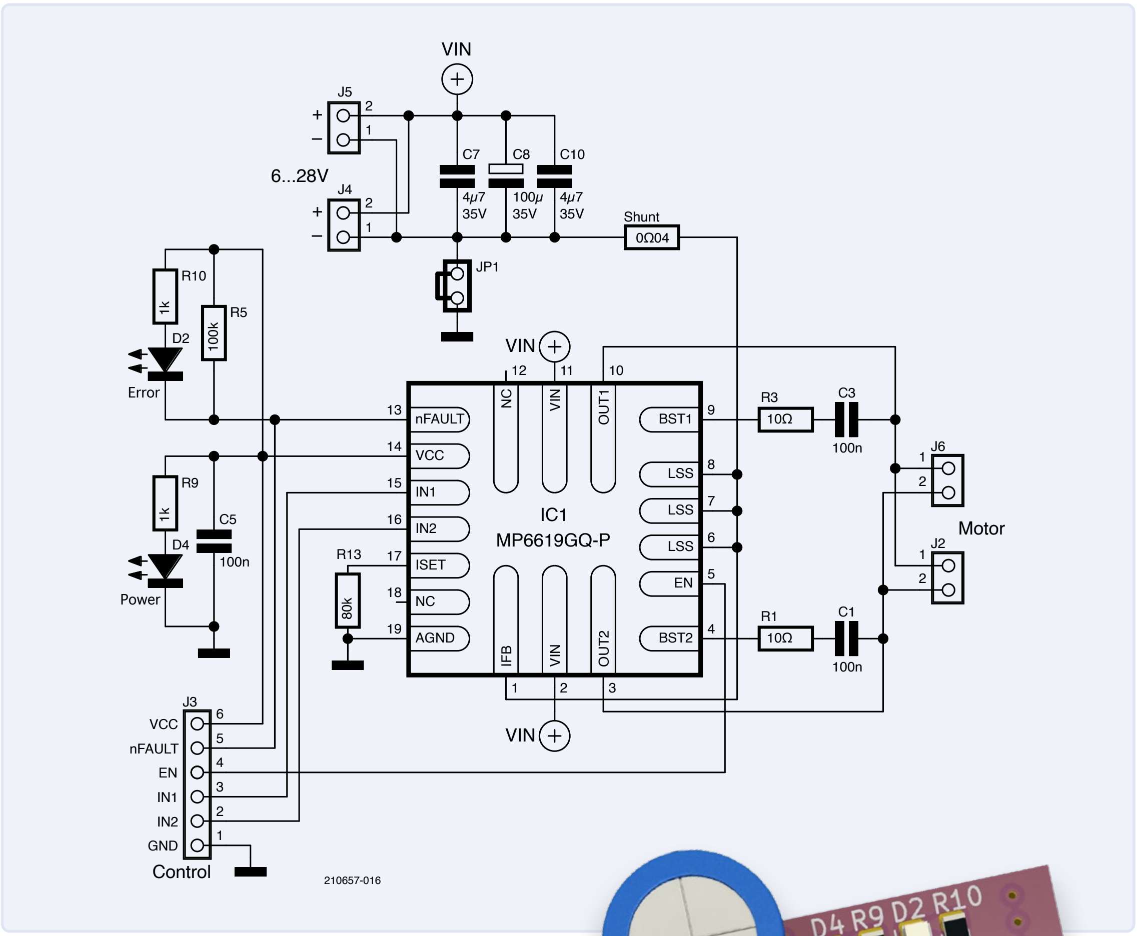


Figure 4: The revised BoB's schematics.

- A clear silkscreen; only the minimal number of components.

This resulted in a new schematic (**Figure 4**) and subsequently also a new PCB (**Figures 5** and **6**) with four layers and an outline of only about one square inch (25×25 mm) produced in a production panel with ENIG finish (**e**lectroless **n**ickel **i**mmersion **g**old). After populating the BoB, I got a really compact module, which, I believe, can be controlled by nearly every microcontroller.

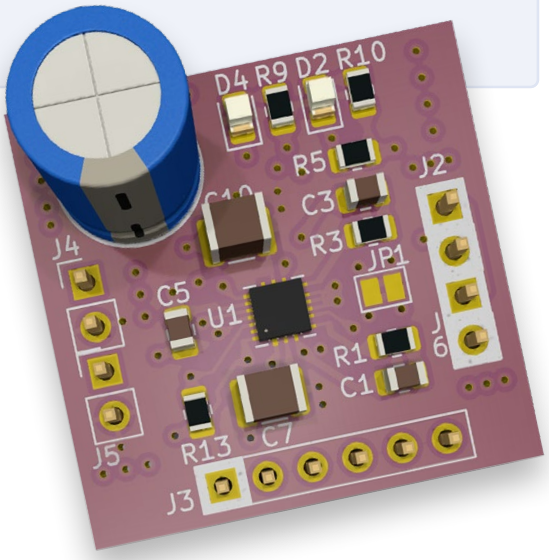


Figure 5: 3D model of the 25×25 mm BoB (top side) consisting of four layers and ENIG finish.

More Tests and Experiments

I used the same motor as in the first test. OVP and ULVO behaved well. Now I tried it with PWM control to realize speed control. Testing with frequencies from 1 kHz to 50 kHz yielded the expected behavior: The motor was nicely controllable. The example from the datasheet uses 20 kHz, and that worked fine at low voltages.

But, at higher voltages, again the OCP kicked in and the motor stopped. By adding an extra 220 µH inductor (from the workbench, like the one from **Figure 7**) in series with the motor, the control behavior was significantly improved, and the speed range was increased.

As a result, the motor delivered more power, and the OCP kicked in less frequently. With these extra

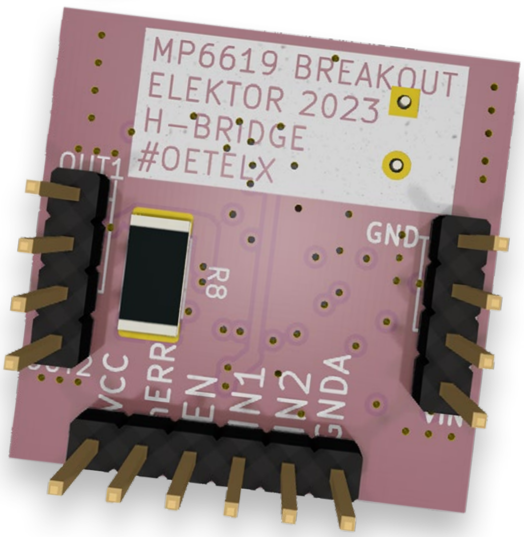


Figure 6: 3D model of the BoB (bottom side) with pin headers.

Figure 7: A toroidal-core inductor in series with the motor improves the BoB's behavior.

Resistors
(SMD 0805)
R1, R3 = 10 Ω
R5 = 100 kΩ
R8 = 0Ω04, 2 W, SMD 2512
R9, R10 = 1 kΩ
R13 = 80 kΩ

Capacitors
C1, C3, C5 = 100 nF, 50 V, SMD 0805

C7, C10 = 4.7 µF, 35 V, SMD 1210
C8 = 100 µF, 35 V, electrolytic, ø 8 mm

Semiconductors
D2 = LED, red, SMD 0806
D4 = LED, green, SMD 0805
IC1 = MP6619GQ-P, QFN-19

Miscellaneous
J2, J4, J5, J6 = 2-pin pin header, 1/10" (2.54 mm)
J3 = 6-pin pin header, 1/10" (2.54 mm)
BoB PCB (see text)

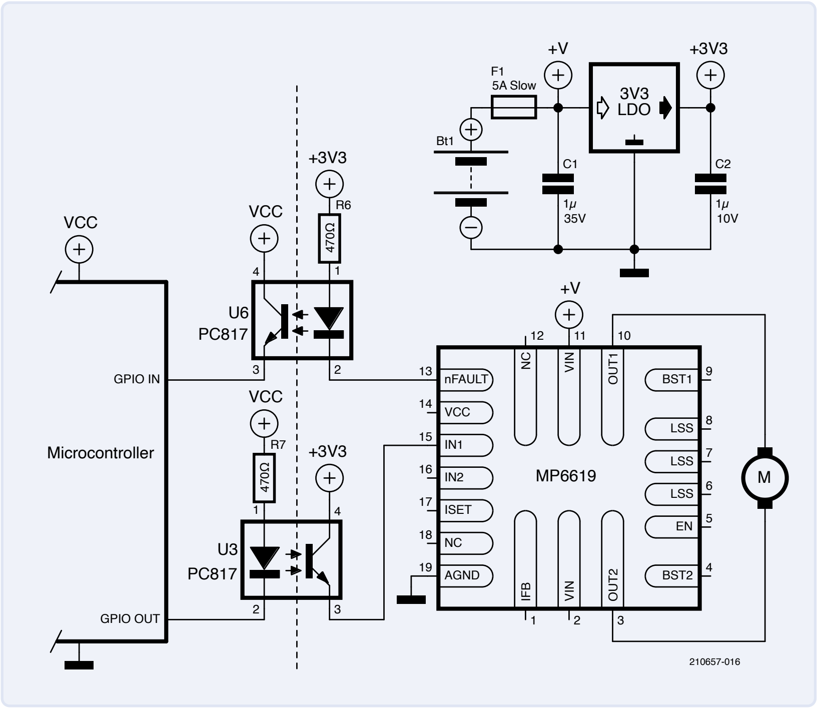
inductors, the solution seems to be suitable for use in projects.

Under some conditions, the motor returns energy, which flows back into the H-bridge. This energy is dumped onto the power rails through the diodes in the MOSFETs, as the motor also functions as a generator (a DC motor's EMF). In my tests, I used low-ESR electrolytic capacitor C8 in parallel to the power source. With this last improvement, the BoB was stable.

Optocouplers

Figure 8 shows my test circuit. I decided to use optocouplers to provide galvanic isolation between the controlling side and the motor drivers, because the power supply of a motor usually carries considerable noise levels. EMI can also cause problems. The isolation reduces the risk of interference in the low-voltage signals and prevents strong pulses generated by the motor and its driver from reaching the sensitive microprocessor and other electronics.

Figure 8: This circuit with optocouplers sitting between the microcontroller and the BoB was used for testing.



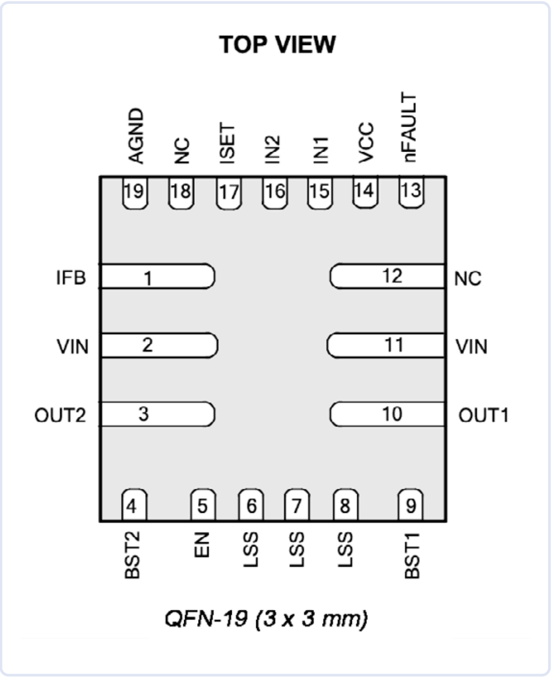
The optocouplers' input LEDs can be driven directly from a microprocessor pin using a 470 Ω resistor in series. On the output side, the phototransitor's emitter is connected to the BoB's IN1, and therefore to the MP6619's input, and its collector coupled to a logic voltage provided by a 3.3 V LDO voltage regulator powered by the motor's power supply — in my case, a battery. This is sufficient to ensure reliable and stable operation.

Figure 8 also shows how you can also bring the H-bridge's nFault signal (Open-Drain output) to the MCU via an optocoupler in the same manner as the other way around. In such a case, the connected microcontroller's GPIO will need a pull-down resistor.

QFN Packages

The FCQFN packages (**Figure 9**) can have irregularly shaped pads, often arranged in long, narrow strips. Unlike regular QFN packages, heat is dissipated through many of these pads instead of one large central pad. This poses some challenges for PCB design since there are many pads, all with different signals, that need to be connected with copper areas.

Small vias can be placed within the pad areas (via in PAD). On multilayer PCBs with power and ground planes, vias can directly connect these pads to the planes. In other cases, copper must be directly attached



◄ *Figure 9: MP6619 QFN-19 package pinout. (Source: Monolithic Power Systems [1])*

to the pads to dissipate heat from the IC to larger copper areas. A layout guideline is available at [4].

The greatest obstacle I encountered during this project was soldering the MP6619 chip. To overcome this problem, I experimented with various techniques, and eventually succeeded with lots of flux, leaded solder and hot air, but it remained a challenge to place the chip precisely.

Once in place, the molten solder's surface tension ensures that the chip is properly aligned with the pads. A lot of practice turned out to be necessary. My conclusion is to purchase a hotplate to simplify the soldering process.

Epilogue

In this article, I shared my experiences in designing and building a breakout board for the MP6619 H-bridge motor driver. I discussed the importance of using optocouplers to provide electrical isolation, and shared my insights into working with flip chip QFN packages and the challenges involved in soldering small components.

The MP6619 breakout board offers a very compact solution for driving low-cost DC motors with brushes, making it a great choice for hobbyists and makers who are building small robots and other creative projects. All design files and production files are open-source and can be found on my GitHub page [5]. The MP6619 is widely available in small quantities from distributors such as Farnell, Mouser, and directly from Monolithic Power Systems.

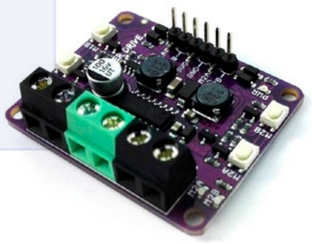
210657-01

About the Author

Edwin van den Oetelaar works as an engineer for Fontys ICT, the Netherlands. He is a member of the High Tech Embedded Software Research Group, led by Prof. Teade Punter. As a specialist in hardware and software development, Edwin relishes the opportunity to tackle diverse technical projects. His proficiencies span a broad array of domains, such as streaming data, smart devices, healthcare, IoT, image processing, and applied AI.

Questions or Comments?

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



WEB LINKS

[1] MP6619 Datasheet @ Monolithic Power Systems: https://tinyurl.com/452pk4sv
[2] First version on Elektor Labs: https://tinyurl.com/299prw8n
[3] DC Planetary Geared Motor: https://tinyurl.com/23nxnr4k
[4] Motor Driver PCB Layout Guidelines — Part 2: https://tinyurl.com/2p8ry9hd
[5] Author's GitHub: https://github.com/edwin-oetelaar

Source: Shutterstock / wk1003mike

From Life's Experience

Hazardous Electronics

By Ilse Joostens (Belgium)

If you regularly order components from large overseas electronics vendors, there's a good chance that, sooner or later, you will encounter a package marked with the notorious "California Proposition 65" warning. In my case this involved PCB connectors, and "Big Brother" kindly advised me that contact with these connectors put me at risk of developing cancer or experiencing fertility problems. It looks like electronics has become the new smoking. I don't take such warnings too seriously, but, even so, if you work with electronics, you do run a risk of exposure to hazardous substances — and I'm not talking about lead solder, solder fumes, flux, etchant, solvents, varnishes, or chemical products with unclear compositions.

A Little Bit Extra

We've all heard that integrated circuits are made from sand. Sand is essentially silicon dioxide, and nowadays silicon is the most commonly used semiconductor material. However, in practice there's a little bit extra in ICs. Not only ICs, but also many other components — semiconductors or not — have a dark side and sometimes contain highly exotic chemical substances.

Arsenic, once used for delightful green wallpaper in Victorian times, is now used in large quantities in the electronics industry for doping silicon and in the production of gallium arsenide, another important semiconductor material for ICs and some transistors. The now-notorious PFAS and PFOS [1] are used in processes for exposing and etching silicon wafers, and residues of these substances occasionally remain in the finished product. Component packages contain not only epoxy, but also flame retardants in the form of organic bromine and chlorine compounds in combination with antimony trioxide [2]. These substances are not highly toxic, but they still have a questionable reputation, so efforts are made to limit their use.

Then, there is beryllium — a real horror show that makes arsenic look harmless. In the old days there was a zinc-beryllium silicate in the white phosphor of fluorescent lamps, until a lot of factory workers came down with berylliosis, a nasty, chronic lung disease [3]. At present, beryllium oxide is still used, among other things as a thermally conductive ceramic material in high-frequency power transistors. Before you have a panic attack: Beryllium copper contacts (often in used as spring contacts) are relatively harmless, as long as you don't file, grind, or sand them.

If you're past a certain age, you most likely remember selenium rectifiers (**Figure 1**) in the form of a stack of thin square plates joined together in the middle. There's no problem,

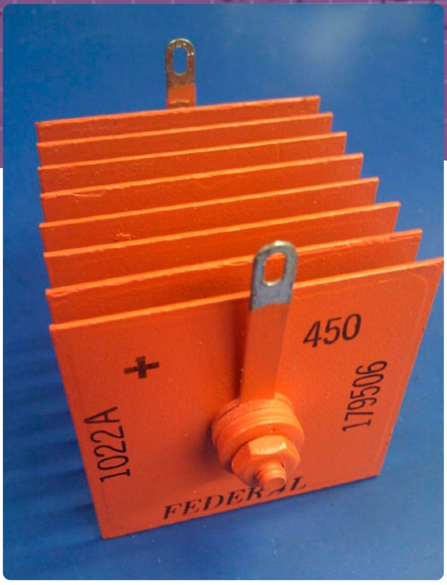


Figure 1: A selenium rectifier. Source: Wikimedia / Binarysequence / https://creativecommons.org/licenses/by-sa/3.0.

as long as things go well, but the smell of an overheated selenium rectifier is something you never forget. The hydrogen selenide released in that situation not only has an obnoxious odor, but is also highly poisonous [4]. Not that anyone has ever died from it — a small mercy.

Another substance in the rogue's gallery is cadmium, best known from nickel-cadmium rechargeable batteries, but also present in LDRs, cadmium-telluride solar panels, sensors, transparent conductors, sometimes in SMD resistors, and even in PVC wire insulation. That means that it's almost everywhere.

Mercury, previously used in batteries, mercury switches, Nixie tubes, energy-efficient lamps, fluorescent lamps, mercury lamps, and mercury-arc rectifiers, is a well-known hazardous substance. Fluorescent lamps are still on the market, and Uncle Ali is your friend if you're looking for mercury switches. Antique octopus-style mercury-arc rectifiers are especially fascinating, even a bit unworldly, with their spooky blue glow (**Figure 2**) [5]. If you break one of them, it's time to call in a HazMat team.

Finally, the phosphors in today's LED lamps contain a diverse range of exotic elements, including the rare earth metals yttrium, cerium, and europium. Although these do not look very poisonous according to their data sheets, the lack of extensive toxicological data is a cause for concern.

Patriotism

Along with the previously mentioned poisonous substances, older components in particular sometimes contain radioactive substances. For example, some tube filaments are coated with materials such as thorium oxide or made of a thorium-tungsten alloy — including the filament of the magnetron in your microwave oven. Uranium can be found in the form of uranium dioxide in the former Urdox PTC resistors (**Figure 3**) and as uranium glass in certain tubes. And then there's a whole list of radioactive vacuum tubes, radar tubes, surge protectors, and gas-filled spark gaps [6]. Just about anything you can imagine — fluorescent lamps, fluorescent lamp starters, metal-hydride lamps, and even Nixie tubes — can contain radioactive material.

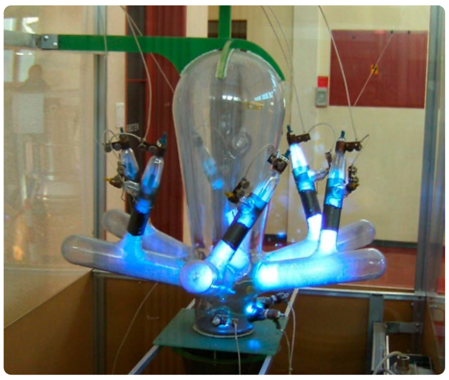


Figure 2: A mercury-arc rectifier. Source: Wikipedia / Timberwind / Public Domain.

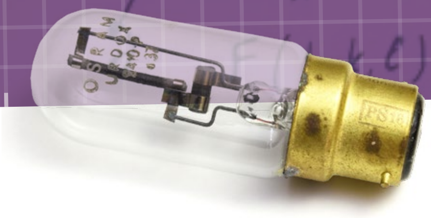


Figure 3: A Urdox PTC resistor. Photo: Ilse Joostens.

Industrial sensors such as thickness gauges, concentration gauges, or sensors for measuring the moisture content of materials also sometimes contain radioactive substances. There's not a big chance you'll come in contact with these devices, but that's not true with the (now-prohibited) ionization smoke detectors. In the early days, there were some types with radium-226, but more recent devices contain americium-241. That reminds me of the man who accidentally smoked the radioactive source from that sort of smoke detector in a cannabis vaporizer. He didn't get high from it, but, according to the story, it had a strange taste [7].

The former Soviet Union also had their own smoke detectors, but, for patriotic reasons, americium-241 was a no-go, so they used reactor-grade plutonium instead, and quite a bit of it — as much as 18.5 MBq in the KI-1 smoke detector [8]. At the time, electronic components were not very sensitive, so a strong source was necessary. Still, it's a strange realization that something like that hung over your head in many government buildings.

After all this, I hope you will still be able to get a good night's sleep. In practice, things are not so bad as long as you don't do something crazy and don't inhale too much magic smoke. However, there is a problem for workers who recycle electronic waste, so let this be a plea for more sustainable electronic products.

Translated by Kenneth Cox — 230411-01

WEB LINKS

[1] PFAS: chips' poisonous ingredient that doesn't go away: https://euractiv.com/section/digital/news/pfas-chips-poisonous-ingredient-that-doesnt-go-away/
[2] Wikipedia: Brominated flame retardant: https://en.wikipedia.org/wiki/Brominated_flame_retardant
[3] Environews Focus: "Beryllium: A Chronic Problem:" https://ehp.niehs.nih.gov/doi/pdf/10.1289/ehp.94102526
[4] YouTube: BigCliveDotCom: Selenium rectifiers - the smelliest components ever: https://youtu.be/OOA1NaoKV6I
[5] YouTube: Kempton Steam Museum: The Mercury Arc Rectifiers: https://youtu.be/YhaQqgXrMMU
[6] YouTube: AE Laboratories: Radioactive Vacuum Tubes: https://youtu.be/m-6-aJv7GCI
[7] YouTube: That Chemist: Somebody Vaped a Smoke Detector: https://youtu.be/14UggnpN39w?t=627
[8] Special Nuclear Material: Analysis of Soviet smoke detector plutonium: https://carlwillis.wordpress.com/2017/02/07/analysis-of-soviet-smoke-detector-plutonium/

Is Cellular the **Lowest-Power Option** for IoT?

LTE-M and NB-IoT Energy Requirements in LPWAN Deployments

By Stuart Cording (Elektor)

When tackling your next IoT project, it's worth casting an eye over the cellular LPWAN offering. While LoRaWAN seems low power on paper, researchers have seen wild variations in battery life with real-life deployments. LTE-M and NB-IoT are both very competitive when it comes to power budgets and offer a range of other benefits on top. But, like all good things, they have their own host of challenges to deal with.

The ability of cellular networks to provide global access to connectivity became clear to me in the early 2000s. I was enjoying being a passenger during a business trip through the Swiss Alps when a colleague called, requesting a presentation for a customer meeting. With my trusty Ericsson T68i balanced on the dashboard, an early Bluetooth-enabled handset, and laptop between my knees, I quickly set about sending the file using the blisteringly fast 115 kbps offered by GPRS. Admittedly, it took several attempts due to losing the connection in the tunnels. Still, the ability to be linked to the world in the middle of nowhere indicated that the era of limitless wireless connectivity had arrived.

First Glimmer of IoT

Thanks to Bluetooth, the phone could also be accessed through a terminal window like the old wired modems that had been replaced by DSL technology. That meant that a Bluetooth-capable microcontroller and some software handling AT commands were all that was required to connect to the world. We were implementing the Internet of Things (IoT); we just didn't know it at the time. Now, 20 years later, IoT is established, and we have even better cellular networks. But, we would probably be hard-pressed to name a product or application we know of that isn't a smartphone or tablet using cellular for data connectivity.

One of the challenges around adoption could be the confusion the entire cellular IoT ecosystem has with its nomenclature, making it challenging to know what to choose and why. At the top level, we're now used to the transition from 4G to 5G, but these marketing terms are only relevant to consumers and businesses for smartphone and high-speed data connectivity. For lower data rate applications used in machine communication, there are separate standards to consider.

Higher Data Rate Cellular IoT

The first is LTE-M, which stands for Long-Term Evolution Machine Type Communication. This branches into two current standards, LTE Cat M1 and LTE Cat M2. The 3rd-Generation Partnership Project (3GPP) defines these standards, with new capabilities ratified in "releases." Thus, LTE Cat M1 was part of Release 13 in 2015, and LTE Cat M2 was part of Release 14 in 2017. Cat M1 offers 1 Mbps uplink and downlink, while Cat M2 offers around 7 Mbps uplink and 4 Mbps downlink. Both support full- and half-duplex (**Figure 1**). For context, 5G smartphone networks average around 100 Mbps [1].

Part of the benefit of a lower data rate is the reduced power demands of LTE-M hardware. According to the 3GPP specification, the aim was to achieve ten years of operation on a 5 Wh battery. However, while that lifetime may be achievable, Brian Ray [2], an engineer now with Google, notes that achieving a 23 dBm transmit power during an uplink, the highest supported level, results in peak currents of around 500 mA. This poses a not-insignificant design challenge.

Coverage is also better than standard LTE thanks to a higher Maximum Coupling Loss (MCL). This figure defines the point at which a wireless system loses its ability to deliver its service. In a study by Sierra Wireless [2], an MCL of up to 164 dB for LTE Cat-M1 was determined, a significant improvement over the 142 dB of legacy LTE and much better than the 155.7 dB target that 3GPP set itself. In terms of capability, this means better connectivity inside buildings, which is important for smart metrology applications where the building can contribute a 50 dB penetration loss, and better range outside.

Like smartphones, LTE-M also supports mobile data, meaning that your device continuously connects to the nearest cell tower, making it ideal for sensors monitoring perishable goods on the move or

	LTE-M		NB-IoT	
	LTE Cat M1	LTE Cat M2	LTE Cat NB1	LTE Cat NB2
3GPP Release	Release 13	Release 14	Release 13	Release 14
Peak Uplink Rate	1 Mb/s	7 Mb/s	66 kb/s	160 kb/s
Peak Downlink Rate	1 Mb/s	4 Mb/s	26 kb/s	127 kb/s
Voice over LTE	yes	yes	no	no
Duplex	full / half	full / half	half	half
Latency	<15 ms	<15 ms	<10 s	<10 s

Figure 1: LTE-M offers mobile IoT with higher data rates and VoLTE, while NB-IoT targets static applications.

tracking fleets of vehicles. Additionally, if you need voice occasionally as part of the system, such as in a fire alarm panel or a monitoring system for the elderly, Voice over LTE (VoLTE) is included. Finally, with a latency of under 15 ms, LTE-M could support an application that seems responsive to a human.

Cellular for Static IoT Nodes

The alternative IoT technology for cellular is NB-IoT. This also comes in two flavors. LTE Cat NB1 was formalized in Release 13, while LTE Cat NB2 has been about since Release 14. NB-IoT targets non-moving applications, such as smart meters in agriculture, weather stations, or sensor deployments at water treatment plants, as it doesn't support cell tower handoff or VoLTE (**Figure 2**). Instead, there is a peak of power consumption as the device registers itself with its nearest tower, after which the wireless module can enter a sleep mode knowing that, upon wake, it can continue from where it left off.

NB-IoT data rates are much lower than those of LTE-M. LTE Cat NB1 achieves 26 kbps downlink and up to 66 kbps uplink, while LTE Cat NB2 reaches 127 kbps in the downlink and around 160 kbps in the uplink. Unlike LTE-M, NB-IoT only supports half-duplex mode. Latency is also much higher, with 1.6 to 10 seconds typically quoted. However, considering the intended use case, responding with a new actuator setting for a greenhouse window or water treatment sluice after receiving some sensor data won't be an issue. It should also be noted that this is much better latency than competing low-power wide area networking (LPWAN) technologies such as LoRaWAN [4] and Sigfox [5].

Achieving Low-Power IoT with Cellular

Power is one of the core requirements for an IoT application, as it will use primarily either batteries or some other renewable energy source, such as a solar panel. Both of these wireless technologies support various low-power modes to give developers options for improving battery life. The first of these is PSM, or Power Saving Mode. This allows the application to place the cellular radio module in a deep sleep state, resulting in a power draw of a few microamps in most cases. The device notifies its cell tower of its intention and can then sleep for up to 413 days. During this time, there is no way to pass data to the device. However, when it wakes up, there is no need to register with the cell tower.

The next mode is eDRX, or extended Discontinuous Reception. This lighter sleep mode saves power for up to 40 minutes with LTE-M or up to three hours with NB-IoT. Waking up from sleep is also faster than PSM. Despite these power-saving levers, they are not always available in all locations. Their configuration depends on the service provider's equipment [6], meaning that battery life in one country could be less than in others because a setting cannot be negotiated. Use of a service provider offering a roaming SIM may also limit access to these low-power features [7].

Because cellular IoT implementation varies so much and is dependent on so many factors, it's no wonder that searching for guidance on power consumption is a fruitless task. More often than not, a Google search delivers pages stating the "10 years of battery life" statement, seemingly for both LTE-M and NB-IoT and without details on battery capacity.

Cellular IoT in the Lab

Thankfully, teams at various institutes have taken the time to study power consumption. Their results provide some guidance on what to expect and show how cellular IoT performs when compared to the alternatives. For example, Tan [8] compares LoRa and NB-IoT. In the experiment, MQTT packets are sent, each containing 50 bytes of data. Using an optimal configuration and good connection conditions, NB-IoT requires around 200 mJ per transaction.

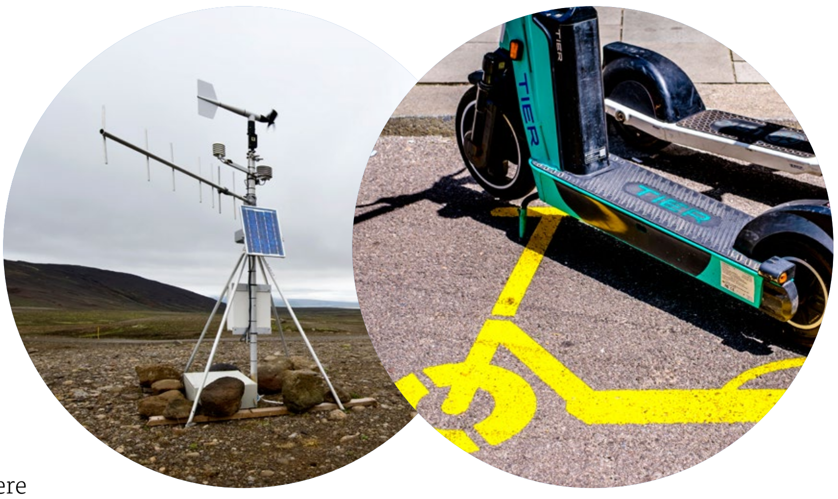


Figure 2: Mobile applications, like e-scooter sharing, are better served with LTE-M. A static application, such as a weather station, benefits from the lower power demands of NB-IoT. (Source: Shutterstock)

LoRa, on the other hand, provides the developer with more control over the transmission configuration by setting a spreading factor (SF). At SF7, the bit rate is higher, meaning less air time, while at SF12, the lowest setting, the air time is the longest when transferring the same number of bytes. The risk, however, is that SF7 reduces the range too much for a successful transfer, meaning the exchange of data must be repeated. During testing, SF7 required just 100 mJ per transaction. However, increasing to SF12 required 250 mJ.

The conclusion is that, for a 3,000 mAh battery, LoRa using SF7 could power this configuration for over 32 years, but using SF12 reduces this to just under 13 years. By comparison, using NB-IoT could deliver just under 20 years of operation. Considering that, under real-life conditions, LoRa would need to adapt its SF and other transceiver configuration (bandwidth) to maintain successful data transfers, such a variation in battery life may be considered too risky.

Comparing with LoRaWAN

This risk is also highlighted in research from a team at the University of Antwerp [9]. In their conclusion, they also state that, while LoRaWAN consumed the lowest power under controlled laboratory conditions indicating an application life of years, "with the real time [sic] deployment it narrowed down to a few months." They also tested NB-IoT alongside Sigfox and DASH7 [10]. While DASH7 offered even better power consumption than LoRaWAN under the same conditions, the team surmised that NB-IoT may still be the better option despite the slightly higher power consumption. NB-IoT may simply tick more boxes when factoring in everything an IoT application requires, such as availability, latency, coverage, security, robustness, and throughput.

Wireless technology has been democratized over the past two decades thanks to CMOS radio transceivers, highly integrated radio modules, and tiny antennas. Even the software stacks are often freely available. However, designers often only work on the end node, relying on others for the infrastructure they'll connect to. Despite LTE cellular networks' ubiquitousness and ease of use for smartphone users, the same cannot be said for those hoping to rely upon it for IoT.

Not for the Faint-Hearted

For the uninitiated, it's challenging finding reliable guidance on what cellular IoT can and can't do, whether important power-saving features are available globally, or how to determine their availability. As such, it makes cellular IoT look like the poor sibling of the smartphone industry. Recent news that Vodafone wants to sell its business focused on IoT services doesn't help shake this image. Although they sold 150m IoT SIM connections last year [11], that division only makes up 2% of their service revenue.

Although research shows cellular IoT to be competitive in power consumption and battery life compared to alternative LPWAN solutions, it still has its foibles. With network infrastructure from some providers and in some countries failing to implement power-saving support universally, engineers will clearly be uncomfortable committing to battery life values that are attractive to customers. What is clear is that each LPWAN approach has disadvantages, and design teams need to weigh them all up based on each use case. And, it would seem a healthy portion of research time is also needed to achieve, as far as possible, an apples-to-apples comparison of the LPWAN solutions that fit the bill most closely.

230376-01

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] "5G vs 4G: What's the difference?," EE Limited, September 2020: https://bit.ly/3MTuUYd
[2] B. Ray, "What is LTE-M?," Medium, May 2017: https://bit.ly/45QDq2U
[3] G. Vos et al., "Coverage Analysis of LTE-M Category-M1," Sierra Wireless, January 2017: https://bit.ly/3OYpoGG
[4] "What are LoRa and LoRaWAN?," The Things Network: https://bit.ly/43EgYc5
[5] Sigfox website: https://sigfox.com/
[6] "PSM and eDRX: Power saving in cellular LPWAN - possibilities and limitations," 1NCE GmbH: https://bit.ly/43p6a10
[7] P. Marshall, "Sleeping Battery: How eDRX and PSM Can Save Energy in LPWA IoT Edge Devices," Eseye, May 2021: https://bit.ly/3Ne1WUh
[8] L. Tan, "Comparison of LoRa and NBIoT in Terms of Power Consumption," KTH Royal Institute of Technology, January 2020: https://bit.ly/43mXrwl
[9] R. K. Singh et al., "Energy Consumption Analysis of LPWAN Technologies and Lifetime Estimation for IoT Application," Sensors (Basel, Switzerland), August 2020: https://bit.ly/3qvfmm7
[10] Website of the DASH7 Alliance: https://www.dash7-alliance.org/
[11] M. Kleinman, "Vodafone dials up sale of stake in £1bn Internet of Things unit," Sky UK, May 2023: https://bit.ly/3MQZUbt

Wireless Communication in IoT Systems Using Arduino MKR Modules

The Right Board for Wi-Fi, LoRa, and Many More Standards

Contributed by Transfer Multisort Elektronik Sp. z o.o.

In this article, we present a brief overview of selected Arduino development kits from the MKR family for rapid prototyping of IoT devices using wireless communication in standards such as WiFi/Bluetooth, LoRaWAN/Sigfox, GSM/3G, or NB-IoT.

One of the biggest problems currently facing the Internet of Things (IoT) [1] device market is its high fragmentation. The multitude of devices and communication protocols makes it very difficult to build a uniform and functional system if you decide to use components from different manufacturers. There are many reasons for this, and they are not always solely related to the designers' desire to push through their own licensed solutions.

The term "IoT" covers many types of devices. These could be, for example, small measurement sensors powered by alternative energy sources and requiring small amounts of data exchange over long distances, but also remote cameras transmitting high-resolution images in real time. Thus, the specificity of these designed devices forces the designers to select the appropriate wireless communication technology for the requirements. You must consider, among other things, battery life, communication range, or amount of data transferred. Responding to the needs of the market, manufacturers of development kits (including those from the Arduino [2] platform) made sure that their portfolios cover the needs of IoT device designers to the fullest extent possible. In this article, we present a brief overview of selected Arduino development kits from the MKR family, prepared for rapid prototyping of IoT devices using wireless communication with standards such as Wi-Fi/Bluetooth, LoRaWAN/Sigfox, GSM/3G, or NB-IoT.

Wi-Fi/Bluetooth: Arduino MKR 1000/1010

Communication in the ISM 2.4 GHz band, using Wi-Fi and Bluetooth standards, has been on the IoT device market for several years. For the rapid implementation of hardware and software prototypes using Wi-Fi communication, Arduino has developed the Arduino MKR WiFi 1000 [3] and MKR WiFi 1010 [4] development kits. The former is based on the ATSAMW25 module [5], containing the SAMD21 microcontroller, WINC1500 [6] radio path and the ECC508 [7] authentication chip. The MKR 1010 kit is fitted with the NINA-W102 radio module from u-blox [8], which offers Bluetooth/BLE communication.

On the software side, Arduino provides the *WiFi101* library, supporting WEP and WPA2 personal encryption for the MKR WiFi 1000 modules. For the MKR WiFi 1010 module (and other kits based on the u-blox NINA-W102 module, including the Arduino NANO 33 IoT [9]) the manufacturer has prepared the *WiFiNINA* library, as well as a number of sample applications demonstrating integration with Android IoT Cloud and Azure, AWS IoT Core, Google Firebase, or Blynk.

LoRaWAN and Sigfox: Arduino MKR WAN 13x0 and FOX 1200 modules

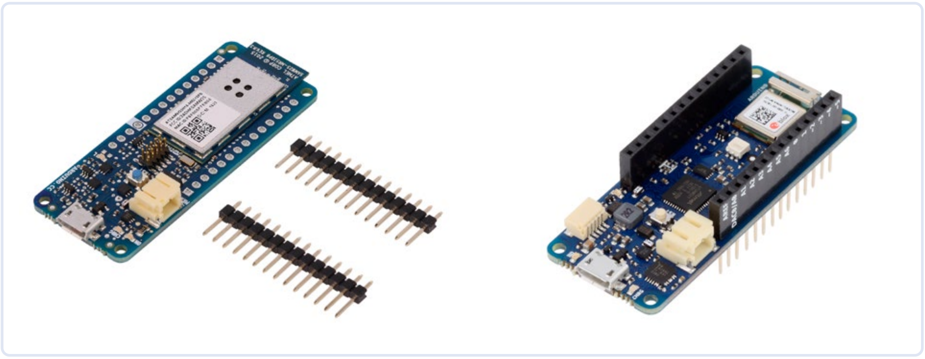
The rapid development of IoT systems has resulted in an increased interest in the subject of smart cities. Unfortunately, Wi-Fi/Bluetooth/BLE standards are used over short distances and do not meet all the requirements set for "Smart City" projects (which include, among others, extensive networks of pollution sensors, water level monitoring sensors, and parking lot sensors). The solution to these problems may be the use of one of the two most popular communication standards in the LPWAN (Low-Power Wide Area Network) area — LoRaWAN or Sigfox, which enable small amounts of data to be transmitted over long distances. For the rapid prototyping of devices using LoRa/LoRaWAN, communication, Arduino designers have prepared the MKR WAN 1300 [10] development kit, and its successor, the MKR WAN 1310 [11]. Both modules are based on the Atmel SAMD21 microcontroller, used in other Arduino MKR series modules, and the Murata CMWX1ZZABZ radio module. The newer version of the module is additionally equipped with 2 MB of flash memory, a new battery-charging circuit, and low-power optimized power supply circuits.

Arduino MKR WAN 13x0 modules work with Arduino IoT Cloud, which is provided by the manufacturer. The complexity of the solutions provided is complemented by the Arduino Pro Gateway LoRa Connectivity [12], optimized for MKR WAN 1310 modules.

An interesting alternative to LoRa/LoRaWAN is the Sigfox standard, which places particular emphasis on communication from nodes to the access gateway. The Arduino MKR FOX 1200 [13] module, based on the Atmel SAMD21 microcontroller, was made available to designers. The Microchip Smart RF ATA8520, tuned to ISM 868 MHz band assigned for Europe, is responsible for radio communication.

GSM/3G: Arduino MKR GSM 1400 module

Even an extended mesh network operating in the LoRa/LoRaWAN technologies is currently unable to provide global coverage. In the case of IoT projects requiring almost unlimited communication range, the best


Figures 1 and 2: Arduino MKR WAN 1000 (left) and MKR WAN 1010 (right) modules.

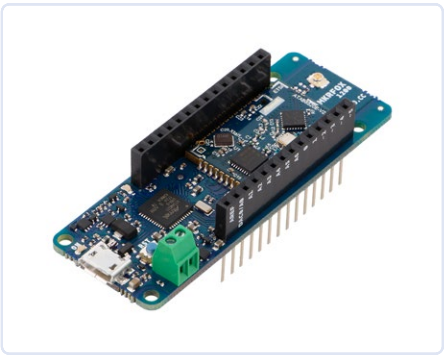
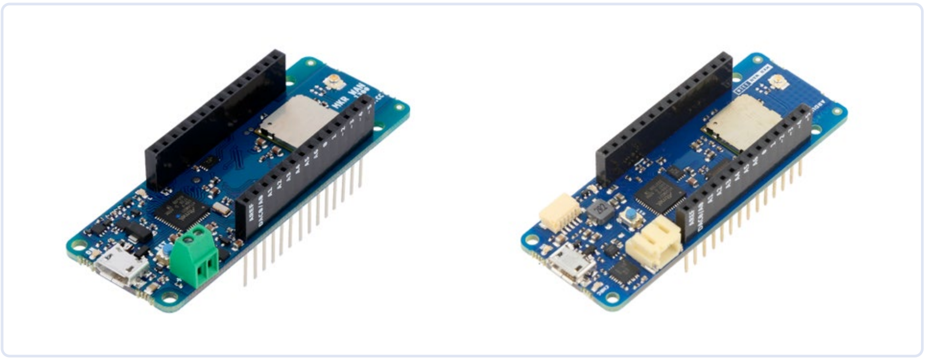


Figure 5: The Arduino MKR FOX 1200 module for communication in Sigfox networks.


Figures 3 and 4: Arduino MKR WAN 1300 (left) and MKR WAN 1310 (right) modules.

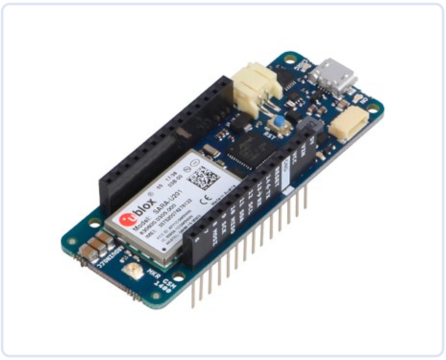


Figure 6: Arduino MKR GSM 1400 module for GSM/3G communication.

solution is to use the GSM/3G standard. Arduino has prepared the MKR GSM 1400 [14] module for GSM/3G communication, featuring a SARA-U210 modem from u-blox and a Microchip ECC508 secure authentication module for the implementation of communication security mechanisms. The built-in GSM modem operates in the following frequency bands: GSM 850 MHz, E-GSM 1900 MHz, DCS 1800 MHz, and PCS 1900 MHz.

To improve the process of software development, the manufacturer provides the *MKRGSM* library (relieving the programmer from operating the module using low-level AT commands), together with a rich set of examples (including GPRS communication, receiving/sending text messages, and handling voice calls). The MKR GSM 1400 module can work with both Arduino IoT Cloud software and alternative cloud solutions: Google IoT Cloud, Blynk, or SORACOM Air IoT, for which the manufacturer has prepared a set of example implementations.

Narrowband IoT: Arduino MKR NB 1500 module:

When briefly characterizing selected communication standards for Internet of Things devices, it is impossible to ignore solutions based on the Narrowband IoT (NB-IoT) standard, which use the licensed LTE 800 MHz band for communication. Similarly to LoRaWAN and Sigfox solutions, NB-IoT belongs to LPWAN networks, thus ensuring stable communication over large areas with the use of energy-saving radio modules that ensure many years of battery operation. Thus, it provides another alternative to LoRaWAN and Sigfox communication in "Smart City" solutions.

For rapid prototyping of NB-IoT end nodes, Arduino has prepared the MKR NB 1500 [15] kit, equipped with the u-blox SARA-R410M-02B [16] module, enabling LTE Cat M1/ NB1 connectivity in the following frequency bands: 1, 2, 3, 4, 5, 8, 12, 13, 18, 19, 20, 25, 26, and 28. In addition, the MKR NB 1500 kit is equipped with Microchip's ECC508 [17] authentication chip, a micro SIM card connector, a Li-Po battery charge controller, and an external antenna connector.

230468-01

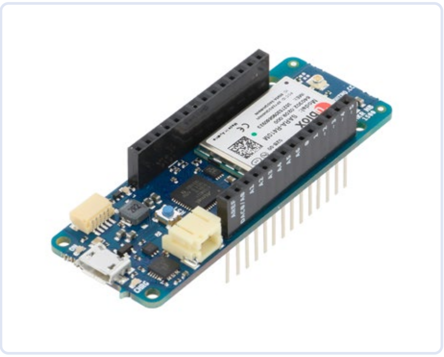
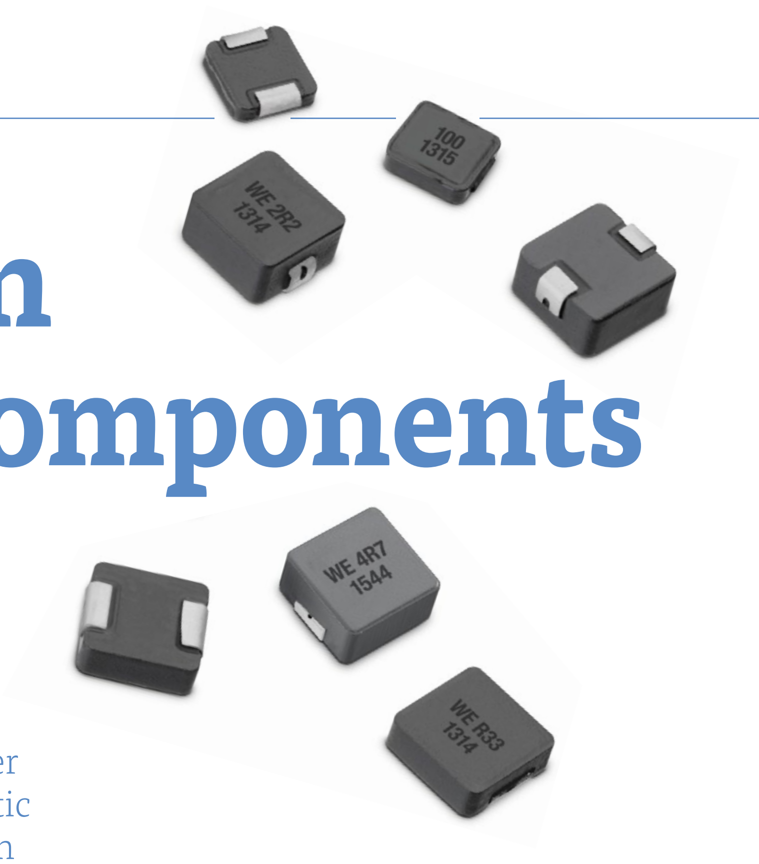


Figure 7: Arduino MKR NB 15 module for narrowband IoT network communication.

Transfer Multisort Elektronik Sp. z o.o.
Headquarters
ul. Ustronna 41
93-350 Łódź, Poland
export@tme.eu
www.tme.eu/en

WEB LINKS

[1] Internet of Things: https://tme.eu/en/katalog/embedded-and-iot-systems_113611
[2] Arduino @ TME: https://tinyurl.com/4b97h9j7
[3] MKR1000 WIFI: https://tinyurl.com/2dxxebu7
[4] MKR WIFI 1010: https://tinyurl.com/4uxfn6sy
[5] ATSAMW25 module: https://tinyurl.com/6dbdmssr
[6] WINC1500 radio path: https://tinyurl.com/45ejhpbn
[7] ECC508 authentication chip: https://tinyurl.com/yx2dvtp7
[8] u-blox: https://tme.eu/en/linecard/p,u-blox_1320
[9] Arduino NANO 33 IoT: https://tinyurl.com/2s3a2jm5
[10] MKR WAN 1300: https://tinyurl.com/4hk3xtjf
[11] MKR WAN 1310: https://tinyurl.com/dra5pske
[12] Arduino Pro Gateway LoRa Connectivity: https://tinyurl.com/4fmenacd
[13] MKR FOX 1200: https://tinyurl.com/2jem6ver
[14] MKR GSM 1400: https://tinyurl.com/ta7rp56s
[15] MKR NB 1500: https://tinyurl.com/yc599x74
[16] SARA-R410M-02B module: https://tinyurl.com/57bvupw4
[17] Microchip Technology @ TME: https://tme.eu/en/linecard/p,microchip-technology_632



AC Losses in Magnetic Components

Avoid Hot Inductors!

By George Slama (Würth Elektronik eiSos)

One of many problems a power supply designer faces is the unexpected overheating of magnetic components that were previously selected with great care. This unexplained behavior can leave the designer confused and frustrated, all of which add pressure to completing their project successfully on time and within budget. Accurate AC loss estimation is a complicated topic and not for the faint of heart. Fortunately, an easy-to-use solution is available.

Despite often looking like simple devices, power supplies are complex projects to design, with many conflicting requirements pressed on by the end user and a multitude of regulatory agencies. The end user wants the least-expensive solution, often in the smallest package, with the highest reliability. The regulatory agencies want minimum thickness insulation and creepage distances, which result in larger components. The designer must have a broad knowledge in many areas besides electrical, mechanical, and control theory. Add to this the mystery of magnetics — where invisible forces act on materials to store or transform energy based on age-old cryptic formulas from the past — and it's no wonder the designer seeks a simpler solution to select the proper magnetic device.

Take the buck regulator as an example, simply because it is the most widely used topology for reducing voltage in non-isolated circuits. The goal of size reduction is realized by increasing the switching frequency. This allows passive components such as inductors and capacitors to be smaller because the energy storage per cycle is lower. Their size is proportional to their energy storage capacity. However, reducing the size reduces the surface area available to dissipate heat from losses, making the thermal design more critical.

In a magnetic device, there are two sources of losses: the windings and the core. Both can be further divided into subtypes (**Table 1**).

What Are Core Losses?

The root of hysteresis core losses lies in the movement of the magnetic dipoles and, at high saturation currents, in the displacement of the core material's domain walls. When a soft magnetic material is influenced by an external magnetic field, caused by a current in a coil or a nearby magnetic field, the magnetic dipoles in the domains (tiny magnetic regions with elementary magnets in the material), align themselves with the field. This takes energy and time. When the outside influence is removed, the magnetic dipoles in the domains reorient and domain walls jump back, but not completely. When the direction of current changes, the magnetic poles in the domains again reverse direction, but not completely. The energy resulting from the spring back is returned to the system, but the rest is expended as work done against the friction of other domains and is converted to heat. The higher the frequency, the more times per second the elementary magnets in the domains and domain areas are moved and shifted, and the more energy is required — exponentially more. The movement of the magnetic dipoles is also proportional to the flux change. Greater flux swing equals more movement equals more energy required, not all of which is recovered. The area within the BH curve represents the energy loss through one cycle.

Eddy current losses stem from the fact that, when alternating currents flow through a conductor, a voltage is induced, per Faraday's law, proportional to the magnetic field's rate of change. The core itself is like a winding, and, although the ferrite material used in high-frequency technology has a high resistance, the small particles conduct. This resistivity characteristic changes exponentially downward as temperature increases. Faster rise times induce higher voltages, and pulses of higher voltage have exponentially greater losses, according to $P = (V^2 \times D) / R$ (where D = duty cycle and R = resistance).

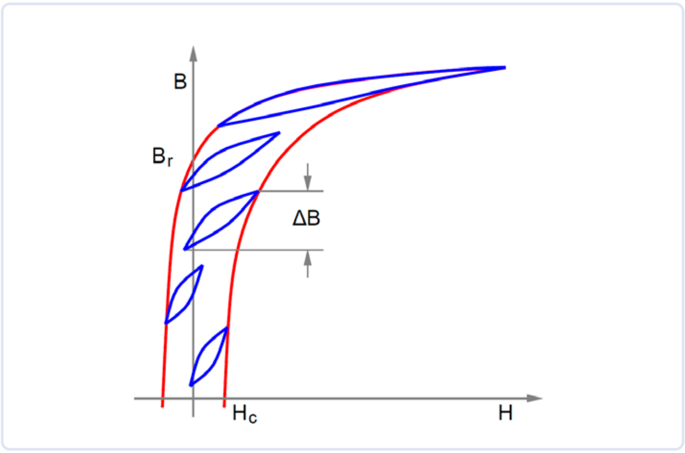


Figure 1: Minor loops are shown at various positions along the major loop. They all have the same peak-to-peak flux swing. The loop area is hysteresis core loss per cycle.

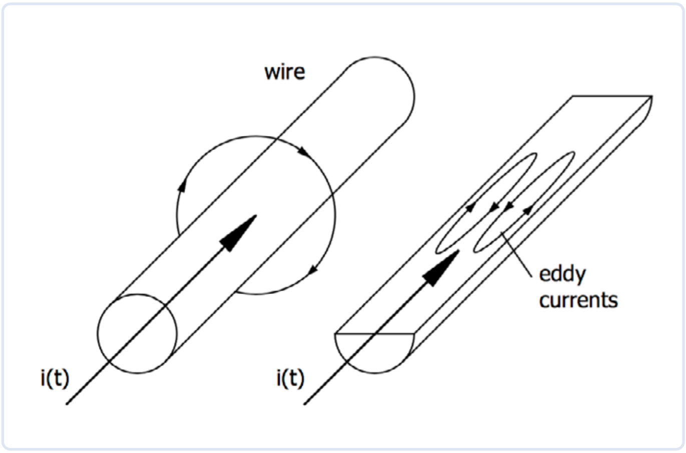


Figure 2: Alternating currents induce a magnetic field, which introduces eddy currents in the opposite direction, cancelling current flow in the center region and reinforcing it in the outer region.

Depending on topology, a third contributor to AC loss is the effect of DC bias. At first, this seems counterintuitive because a static current does not cause any movement of the magnetic dipoles in the domains beyond the initial change. However, when an AC waveform is DC-biased (as a DC current offset of an alternating current), the minor hysteresis curve changes its shape as a function of the polarity and amplitude of the current at different positions along the BH curve (**Figure 1**). Measurements show that, at low bias levels, the influence is small, but at higher levels there is a significant increase in losses. It is understood that, as the magnetic material approaches saturation, when almost all magnetic dipoles are aligned, more energy is required to align the remaining domains' magnetic dipoles. Traditional methods for calculating AC core loss do not account for this, adding further to an unexpected surprise.

Finally, excess losses are attributed to the difference between calculated losses and measured losses. These result from things such as relaxation effects, excess eddy currents, stray losses, and other less-understood phenomena.

What Is Winding Loss?

DC winding loss comes from the DC resistance of the conductor used for the winding. This is simply the measured DC resistance multiplied by the DC current portion of the current waveform squared: $P = I^2 \times R$. The AC winding loss consists of skin and predominantly proximity losses from the AC portion of the current waveform.

Table 1. Losses sources, types, and influences.

Source	Current type	Loss type	Influenced by
Core	AC	Hysteresis	Core material, temperature, waveform, permeability
		Eddy currents	Applied voltage, duty cycle, resistivity, permeability, permittivity
	DC bias	(Hysteresis)	Shifts hysteresis curve, distorts minor loop, topology-dependent
		Excess	Unexplained losses*
Winding	DC	Resistive	Material, temperature
	AC	Skin effect	Frequency, waveform (harmonics), temperature, position*
		Proximity effect	Frequency, waveform (harmonics), position, number of layers, temperature, leakage flux, layering of windings (interleaving)

** Explanation in the text*

This *skin effect* is because, at high frequencies, the current density is no longer constant across the conductor cross-section because the current is displaced toward the surface of the conductor due to finite inductances. The changing current in the conductor creates a magnetic field around the wire, again according to Faraday's law, but which, by Lenz's law, induces a current back into the conductor in the opposite direction. These eddy currents cause cancellation at the center and reinforce currents near the outer surface (**Figure 2**). Skin effect, as commonly defined, is only valid for a single conductor in free space far from other conductors. This is not the case in inductors or transformers, where there are normally many turns and layers of wire tightly wound together.

Proximity effect describes the influence of adjacent magnetic fields on the currents in a conductor. There are two effects: Adjacent wires with currents flowing in the same direction will attract each other (the fields between them cancel) leaving the facing surface areas with little current, as in **Figure 3a**. Adjacent wires with opposing currents will repel each other (fields between add together) and the facing surfaces will have a greater concentration of current, with the opposite sides having considerably less (see **Figure 3b**).

In transformers, the current is in the same direction within a winding but in the opposite direction between primary and secondary windings. Inductors with one winding have current only in the same direction. One of the most important influences in AC winding losses is the number of layers. This is especially prevalent with inductors because, with each layer, the MMF (magneto motive force)

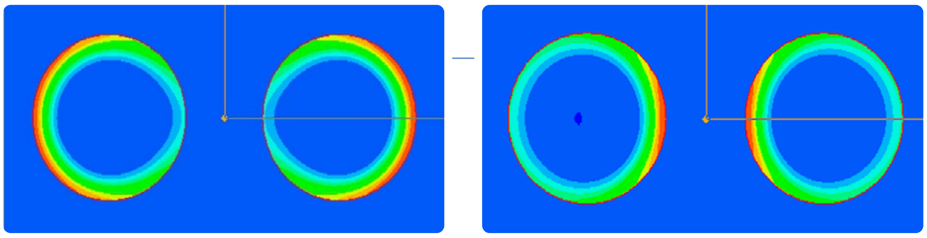


Figure 3: Currents in adjacent wires. Left: with current in the same direction. Right: with current in the opposite direction. Red is high current density; blue is low density.

increases and is not cancelled by a secondary winding. With each additional layer, the losses increase exponentially, since the magneto motive force is the product of the magnetic flux and the magnetic resistance (reluctance).

It's clear to see why single-layer, edge-wound flat wire inductors (aka helical wound coils) have become so popular in high-current inductors. The current, whether AC or DC, will still crowd around the path of least resistance, which is the inside diameter, but the added proximity losses from multiple layers is gone.

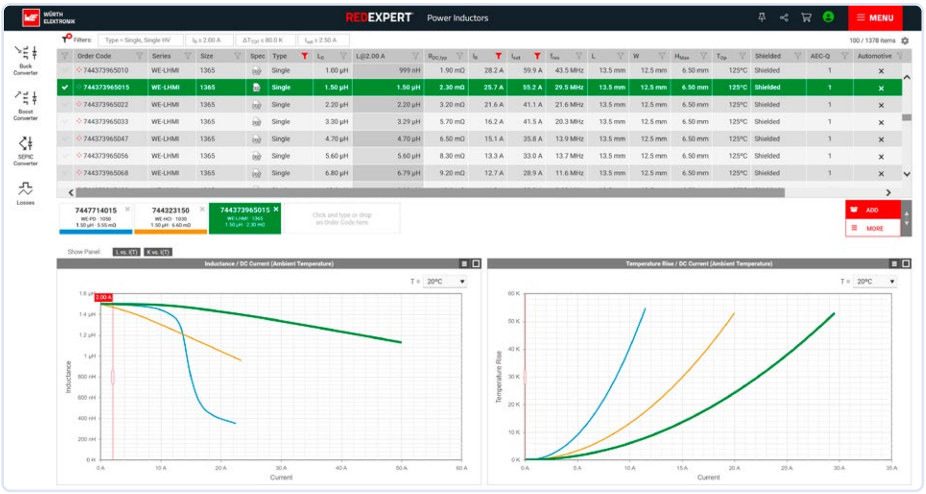
Inductor Selection With REDEXPERT

Over the years, Würth Elektronik eiSos has taken thousands of loss measurements of inductors in its portfolio under real-use conditions — rectangular waveforms, duty cycle, DC bias, ripple current, and temperature. This rich data set of total AC losses includes all the effects of construction methods, wire types, core material, and excitation waveform. No need to do tedious and complicated calculations, often with missing information, or to try modelling losses with arrays of resistors, inductors, and capacitors in order to run simulations for each possible choice.

The Würth Elektronik eiSos online inductor selection tool, REDEXPERT (**Figure 4**) provides convenient access to this data with the ability to compare multiple inductors instantly. Simply select the converter type, enter the basic operating conditions, and all usable inductors are presented. From these, sort and select down to a list of inductors that meet your requirements, including size, height and shape. Easily adjust operating conditions to check the extremes of your design. Charts immediately give the full performance range, including temperature effects. This allows the user to compare the merits of each inductor to their own satisfaction. Then download only the most suitable inductor's file, and, at the same time, you can order free samples.

REDEXPERT will save your work automatically when you click on the share icon in the top menu bar, where a unique URL will be presented. Save it to your design book, or email it to yourself or a colleague to share. The exact display you see will be reproduced when you need it again.

230470-01



Filters: Type = Single, Single HV | $I_R \geq$ 2.00 A | $\Delta T_{(IR)} \leq$ 80.0 K | $I_{sat} \geq$ 2.50 A

100 / 1378 items

	Order Code	Series	Size	Spec	Type	L_0	L@2.00 A	$R_{DC,typ}$	I_R	I_{sat}	f_{res}	L	W	H_{Max}	T_{Op}	Shielded	AEC-Q	Automotive
	744373965010	WE-LHMI	1365		Single	1.00 µH	999 nH	1.90 mΩ	28.2 A	59.9 A	43.5 MHz	13.5 mm	12.5 mm	6.50 mm	125°C	Shielded	1	×
✓	744373965015	WE-LHMI	1365		Single	1.50 µH	1.50 µH	2.30 mΩ	25.7 A	55.2 A	29.5 MHz	13.5 mm	12.5 mm	6.50 mm	125°C	Shielded	1	×
	744373965022	WE-LHMI	1365		Single	2.20 µH	2.20 µH	3.20 mΩ	21.6 A	41.1 A	21.6 MHz	13.5 mm	12.5 mm	6.50 mm	125°C	Shielded	1	×
	744373965033	WE-LHMI	1365		Single	3.30 µH	3.29 µH	5.70 mΩ	16.2 A	41.5 A	20.3 MHz	13.5 mm	12.5 mm	6.50 mm	125°C	Shielded	1	×
	744373965047	WE-LHMI	1365		Single	4.70 µH	4.70 µH	6.50 mΩ	15.1 A	35.8 A	13.9 MHz	13.5 mm	12.5 mm	6.50 mm	125°C	Shielded	1	×
	744373965056	WE-LHMI	1365		Single	5.60 µH	5.60 µH	8.30 mΩ	13.3 A	33.0 A	13.7 MHz	13.5 mm	12.5 mm	6.50 mm	125°C	Shielded	1	×
	744373965068	WE-LHMI	1365		Single	6.80 µH	6.79 µH	9.20 mΩ	12.7 A	28.9 A	11.6 MHz	13.5 mm	12.5 mm	6.50 mm	125°C	Shielded	1	×

7447714015 — WE-PD · 1050 — 1.50 µH · 5.55 mΩ
744323150 — WE-HCI · 1030 — 1.50 µH · 6.60 mΩ
744373965015 — WE-LHMI · 1365 — 1.50 µH · 2.30 mΩ
Click and type or drop an Order Code here

Figure 4: Tables and charts in REDEXPERT allow for quick and accurate comparison of inductors.

About the Author

George Slama has been designing and working with transformers over his entire 40+ year career. His design experience extends from milliwatt audio and telecom transformers to ferro-resonant, radar, high voltage, small three-phase, high-frequency switching and LTCC transformers and inductors. His work has included quality control, automated testing and manufacturing engineering as well as all aspects of custom switch mode power supply design and development. George has given numerous magnetics design seminars at conferences in the US and Europe. Currently, he is working as a senior application and content engineer at Würth Elektronik, developing application notes and tools to help power supply designers solve their magnetics challenges.

WEB LINKS AND LITERATURE

[1] Baguley, C., Carsten, B. Madawala, "The Effect of DC Bias Conditions on Ferrite Core Losses," IEEE Transactions on Magnetics, Vol. 44, No. 2, February 2008.
[2] Barbisio, E., Fiorillo, F., Ragusa, C., "Predicting Loss on Magnetic Steels Under Arbitrary Induction Waveform and With Minor Hysteresis Loops," IEEE Transactions on Magnetics, Vol. 40, No. 4, July 2004.
[3] Online simulation platform REDEXPERT: https://redexpert.we-online.com

Measurement for Optimal Cloud Deployment

By Stuart Cording, for Mouser Electronics

With the advent of the Internet of Things (IoT) and Industry 4.0, traditional embedded systems have been augmented by cloud technology. Real-time computing, such as that used to control a robot arm or conveyor belt, is linked to servers that collect and evaluate usage data. With essentially unlimited storage and masses of processing power, predictive maintenance, energy-use analysis, and other big-picture assessments can be made — something that was, previously, challenging to implement at scale. These systems often use machine learning technology to analyze the data and make predictions about trends. While much of this may still seem like science fiction, many projects are underway to investigate the technical challenges. One example is Audi, who, together with Ericsson, demonstrated real-time robot arm control using the ultra-reliable low-latency communication (URLLC) capability of 5G for factory automation [1].

Site Surveys for Wireless Connectivity

Running cables for connectivity is costly and challenging, and it fixes the installation point of equipment. This goes against the current trend for manufacturing flexibility, which foresees factory layouts adapting to new use cases, requiring wireless connectivity. In other applications, such as autonomous guided vehicles (AGV), wireless is the only option. Before the rollout of such systems, it is prudent to survey the site to understand the telecoms landscape properly. While telecoms operators provide coverage maps, the reality on site can differ significantly, especially inside a building.

Test equipment such as the Siretta SNYPER-LTE Spectrum (EU) [2] places a high-performance portable spectrum analyzer in the hands of users, simplifying site surveys (**Figure 1**). Targeting EU mobile networks, the device comes in a practical carry-case with everything neatly placed in foam sections. Besides the spectrum analyzer, the kit includes a multi-regional mains and car charger, and USB and RF cables. Two short antennas are included, one for general-purpose 4G/3G/2G, and a second dedicated to 2600 MHz LTE. A third directional antenna supports directional liveSCAN measurements. Users need to provide an activated 4G-capable SIM to perform measurements.

The SNYPER-LTE Spectrum can perform single-point surveys of a location to collect the network's characteristics, including cell ownership and signal strength. The integrated battery provides operation for 48 hours of individual surveys (assuming 20 surveys per day) or up to 15 hours of continuous operation for a liveSCAN survey. The device also has space for the results of 50 surveys.

liveSCAN measurements build upon single-point surveys to generate more detailed analysis. The spectrum analyzer can be

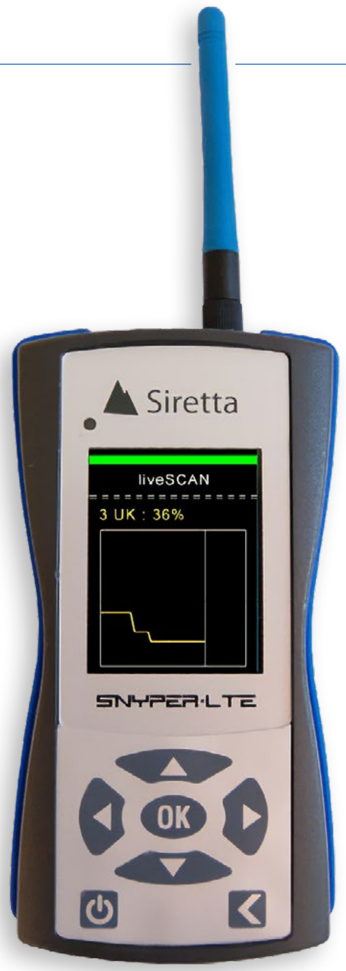


Figure 1: The SNYPER-LTE Spectrum delivers a wealth of data that eases antenna selection and optimal location for cellular wireless equipment.

moved within the building using the standard antenna to determine hotspots with optimal signal strength. Switching to the directional antenna enables the user to find the direction of the highest signal strength. Optimal antenna choice and the location for wireless cellular equipment is then defined using this data. Results are provided on-screen or can be downloaded in comma-delimited (CSV) and HTML formats for reporting and documentation.

Test System Development in the L-Band

Complex RF test systems are increasingly simpler to develop using software-defined radio approaches (SDR), thanks to their high-quality RF front-ends coupled with configurable hardware, such as FPGAs. The BittWare RFX-8440 [3] is a radio frequency system-on-chip (RFSoC) data capture card that targets applications in the L-Band (1 – 2 GHz), used by applications as diverse as GPS, telecommunications, aircraft, and astronomy.

Figure 2: The RFX-8440 provides a highly configurable L-Band capture card suitable for RF test and measurement or volume deployment applications.

The capture card (**Figure 2**) couples a Xilinx Zynq UltraScale+ ZU43 RFSoC with an analog front-end featuring low-noise, variable gain (-40 to 0 dBm) signal conditioning prior to the digitizers. Alternative front-end configurations are also available, extending the input range to 4 GHz. In total, there are four 14-bit, 5 GS/s ADCs, and four 14-bit, 10 GS/s DACs, complemented by clock, reference, and trigger signals. The RFSoC, in addition to its programmable logic, offers a quad-core Arm Cortex-A53 and a dual-core Arm Cortex-R5. Although the capture card features PCIe, it can also be used standalone thanks to the Ethernet, USB, and Display Link interfaces. Further expansion is possible via the eight-channel OCulink port that offers PCIe Gen4 ×8, a connection for NVMe storage, or dual, 100 Gbit networking.

Test systems typically require programmable switches, allowing RF signals to be routed to the correct ports. Teledyne's programmable RF multiplexer T3SP-D4MX-BUNDLE [4] complements the BittWare capture card with a DC to 10 GHz bandwidth- and phase-matching. The device comes with a pair of phase-matched (±2 ps), 8-inch (20 cm) cables for the inputs, and four pairs of phase-matched, color-coded 24-inch (60 cm) cables for the outputs. The two channels each offer 1:4 switches plus a not-connected option that functions as the default selection at power on. Via a USB 2.0 interface, the multiplexer can be controlled using a DLL in C/C++, C#, and Python, or integrated into development environments such as LabVIEW and MatLab. A simple software utility is also available for manual control (**Figure 3**). Each channel offers 1 billion switching cycles thanks to RF microelectromechanical systems (MEMS) switches.

Getting 'Cloud-Ready'

Of course, there is still plenty of untethered machinery in use, free from cloud integration. However, because data analysis enables optimized decision-making, many manufacturers wish to fit data capture solutions retrospectively. With eight channels of analog inputs, the DFRobot DAM-3918 [5] can be used to share analog sensor data over its optically isolated RS-485 interface using the Modbus RTU protocol. With an accuracy of ±1%, it is suitable for use with 2, 3, and 4-wire transmitters and transducers (e.g., 0–5 V, 4–20 mA). Each of the 12-bit input channels is configurable separately, with 50 S/s for a single channel and 400 S/s available in total. Power must be provided from an 18–30 VDC source.

Oscilloscopes and Remote Data Acquisition

With powerful laptops widely available and in regular use by development engineers, test and measurement vendors have introduced a broad range of compact desktop oscilloscopes that rely on your computer's screen for displaying measurements. This places high-performance analog front ends into the developer's hands for minimal outlay, while integrating a range of advanced analysis and protocol-decoding features. Digilent's Analog Discovery Pro 3000 [6] is just such a measurement tool, available in four-channel (ADP3450) and two-channel (ADP3250) options.

In addition to the 14-bit, 0.5 GS/s analog inputs are 16 digital I/Os and a programmable digital power supply. Together with Digilent's *WaveForms* software [7], the Discovery Pro 3000 also offers waveform generation, spectrum analysis, and operation as a network analyzer, to name but a few features. Each instrument can also be controlled using code written in JavaScript or via the WaveForms software development kit (SDK) in C/C++, C#, Visual Basic, or Python.

A unique feature is its Linux Mode, with which the unit operates standalone as a powerful hardware test solution. Coupled with the WaveForms SDK, this terminal-based mode enables the creation and programming of custom tests or applications. The results can be streamed via its Ethernet interface or stored in the local capture buffers, where there is space for millions of data points.

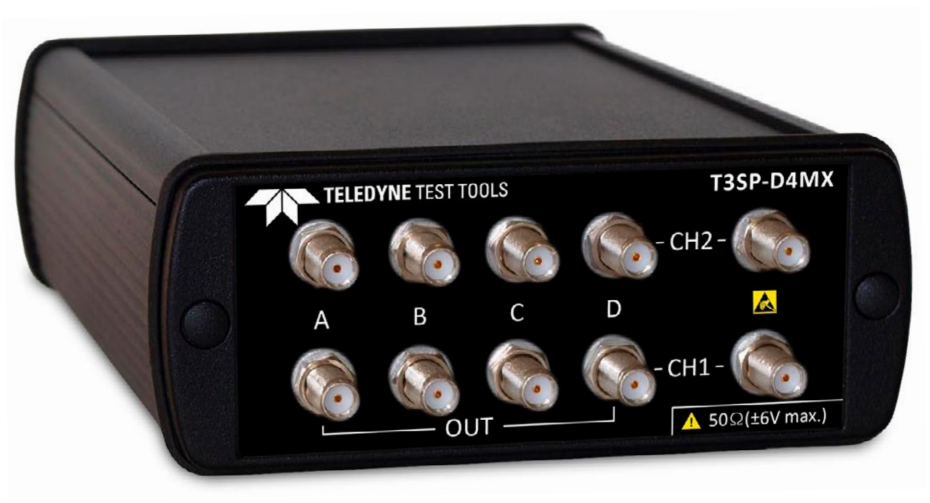


Figure 3: The Teledyne T3SP-D4MX provides a two-channel, DC to 10 GHz 1:4 multiplexer controllable via a DLL or the *WinD4MX.exe* GUI.

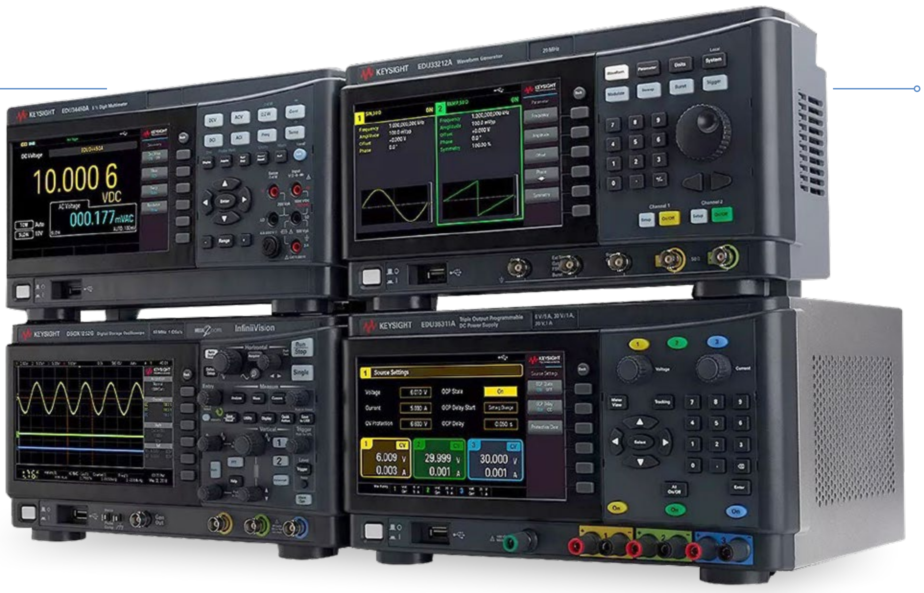


Figure 4: Designed for remote operation and collaboration, Keysight's Smart Bench Essentials *test and measurement equipment unites traditional operation with cloud connectivity.*

The Smarter Bench

While laptop-connected test equipment can offer a lot in a small package, it clutters the bench. It also makes it more challenging to use the computer for email, report writing, and the inevitable video call. Keysight Technologies' Smart Bench Essentials [8] is a family of traditional, stackable, test and measurement tools with industry-grade connectivity, well-dimensioned 7-inch (18 cm) displays, and measurement-sharing capabilities.

The family consists of two- and four-channel oscilloscopes, a 5½ digit multimeter, one and two-channel arbitrary waveform generators, and a triple-output programmable power supply (**Figure 4**). The required bench real estate can be minimized using the instrument stacking kit. The BenchVue software provides data logging and analysis. However, the main attraction is the PathWave Lab Management software. Managers and educators can keep track of equipment connected via wired Ethernet or Wi-Fi, roll out firmware updates, or make the measurement tools available remotely via the cloud.

Test and Measurement for Cloud Deployment

Cloud connectivity continues to increase in the markets engineers address, requiring an on-site understanding of the wireless landscape, advanced and configurable RF test approaches, or solutions that can cloud-enable older, still-useful manufacturing equipment. Display-less oscilloscopes, thanks to their programmability, can be repurposed as highly versatile, remote test and data logging tools. These, too, are capable of streaming their measurements to cloud platforms for further analysis. Finally, even traditional, one-instrument-per-box test equipment offers advanced and robust connectivity, enabling efficient lab management, measurement sharing, and cloud-based control. It seems that, whichever way you look, there is a cloud approach available for test and measurement.

230458-01

Mouser Electronics
Hillbottom Road
High Wycombe
HP12 4HJ
United Kingdom
+44 (0) 1494-427500
uk@mouser.com

About the Author

Stuart Cording is a freelance journalist who writes for, amongst others, Mouser Electronics. He specializes in video content and is focused on technical deep-dives and insight. This makes him particularly interested in the technology itself, how it fits into end applications, and predictions on future advancements.

Mouser Electronics is an authorized semiconductor and electronic component distributor focused on new product introductions from its leading manufacturer partners.

WEB LINKS

[1] 5G for factory automation: https://ericsson.com/en/news/2020/2/5g-for-factory-automation
[2] Siretta SNYPER-LTE Spectrum: https://bit.ly/46DLXqf
[3] BittWare RFX-8440 Card: https://eu.mouser.com/new/test-measurement/bittware-rfx-8440
[4] T3SP-D4MX-BUNDLE: https://bit.ly/3D0q476
[5] DFRobot DAM-3918: https://bit.ly/3D3gPTC
[6] Digilent Analog Discovery Pro 3000: https://bit.ly/3PR8XMR
[7] Digilent WaveForms : https://digilent.com/shop/software/digilent-waveforms
[8] Keysight Smart Bench Essentials: https://bit.ly/3JRekaP

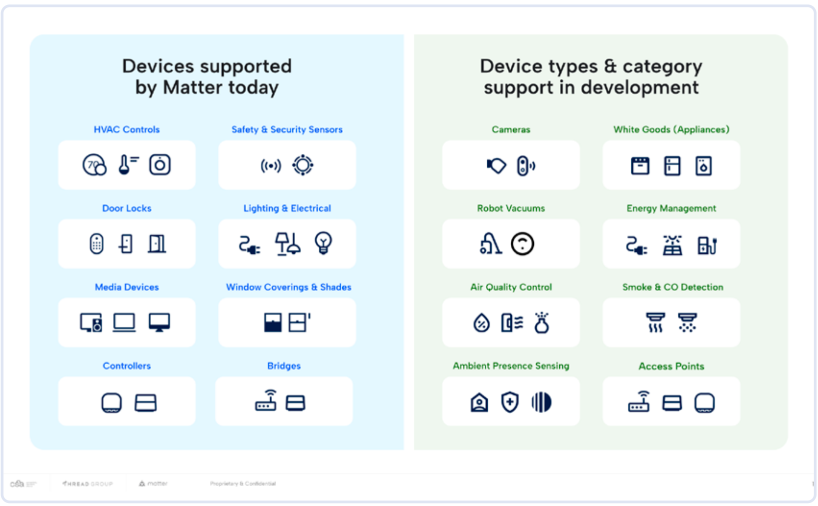
Matter Adoption: What does it take to deploy Matter devices?

By Sujata Neidig, NXP Semiconductors

Owned by the Connectivity Standards Alliance (CSA), Matter is a universal and open IoT protocol, a common language, that enables smart home devices to work with each other across brands and smart home platforms, e.g., Amazon, Apple, Google, Samsung SmartThings, etc. Matter removes walled gardens and brings interoperability so that consumers can have flexibility and choice in the devices they purchase and confidence that they will work together seamlessly. For a deeper dive into Matter, see the *Matter — Making Smart Homes Smarter* paper [1].

Matter also raises the bar on security. Every Matter device must be able to prove its identity and authenticate that it is a Matter certified device before it's allowed to join the Matter network. Once the device is on the network, all communications are encrypted. For a deeper dive into Matter security, see the *Matter — Making Smart Homes More Secure* paper [2].

Figure 1: Matter's expanding device categories (Source: Connectivity Standards Alliance - CSA).



Matter Device Types

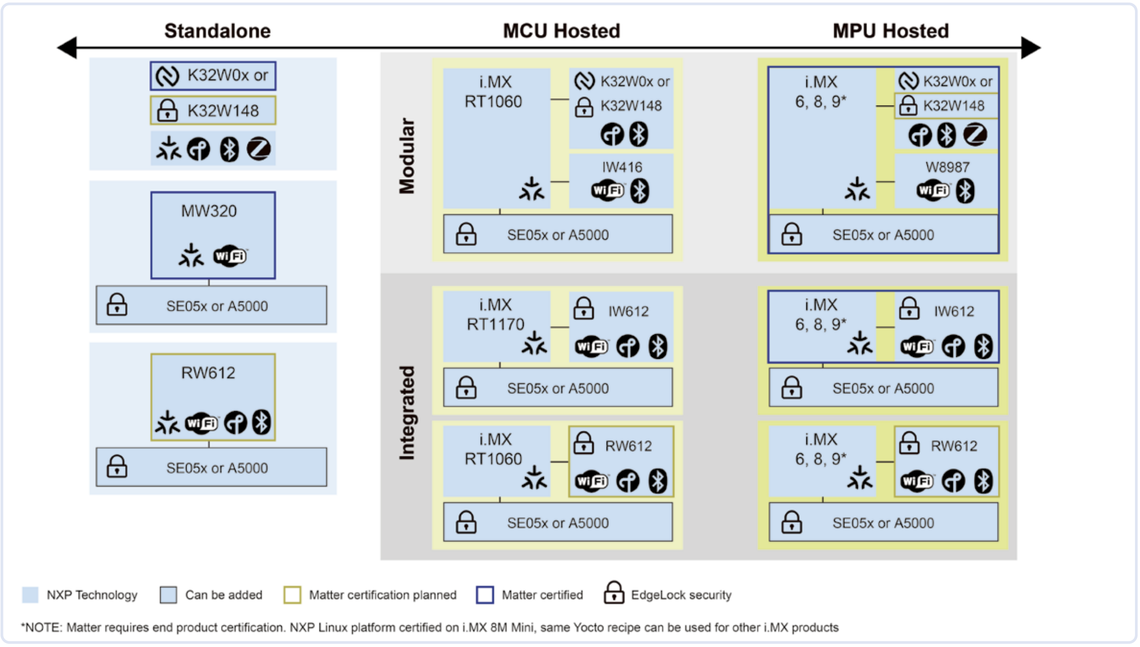
Matter is how devices communicate with each other, it defines the characteristics and capabilities of devices at the application level. At launch, Matter supported seven categories of devices and will expand to many more, as seen in **Figure 1**. CSA members drive expanding Matter into more device types.

Design Considerations

Matter presents developers with several considerations to evaluate in order to architect their system, select the components and plan how to implement security protections.

First, a developer must decide what functionality the device needs based on the Matter device type, associated application clusters, and features needed beyond Matter. For example, the user interface and power envelope targets (e.g., mains or battery powered, battery size and life, etc.). Next, the developer should determine the connectivity requirements. As an IP-based technology, Matter currently supports Wi-Fi, Thread, and Ethernet, and devices can use one or more of these options. Wi-Fi is ideally suited for high bandwidth use cases, such as streaming audio or video, while Thread is ideally suited for low bandwidth control use cases where reliability and low-energy usage are a priority. In addition to connectivity, the developer must determine what role or roles the device will support such as Thread Border Router, Matter Commissioner, Matter Controller, Matter Bridge, etc. The final step is to assess the application's security requirements above what Matter defines. For example, a smart door lock might implement protections against physical attacks.

Once these requirements are mapped out, it's time to determine the system architecture (**Figure 2**):



*NOTE: Matter requires end product certification. NXP Linux platform certified on i.MX 8M Mini, same Yocto recipe can be used for other i.MX products

Figure 2: System architecture options for Matter devices: Standalone and Hosted. (Source: NXP Semiconductors)

- A single MCU is used to implement the application and wireless connectivity. This is an ideal architecture for simpler device types that are cost, size, and power sensitive.
- A host MPU or MCU is used to implement the application with a separate wireless MCU or transceiver that has the hardware for the radio. This architecture is used for more complex device types that have richer user interfaces and/or are managing multiple networks and roles.

NXP provides a portfolio of Matter Development Platforms to address the range of device types and use cases. These platforms include the key components needed for the overall system: processing, connectivity and security. Visit NXP's website [3] for details.

Deploying your Matter Device

Certification is the next step after designing the device, which is an essential to delivering interoperability. Additionally, certification provides the license grant to use the technology royalty-free and the rights to use the technology badges. The CSA provides a certification program for Matter which includes test scripts, tools and services (via Authorized Test Labs) to aid in the process including the Dependent Certification requirement which validates certification of the underlying technologies used – Thread, Wi-Fi and Bluetooth. Note: These technologies are owned by other standards bodies which each provide membership options.

Since Matter is at the application layer, every Matter device must go through Matter certification. Pre-testing can be done by the developer using tools provided by CSA. Once ready, the developer takes the device to an ATL and applies for certification to CSA. CSA will confirm the Matter certification, the dependent certifications and issue the certification ID. The device is then added to the CSA's certified products listing and Distributed Compliance Ledger.

Since Thread is a network layer that is usually not altered by the application, Thread Group's certification program supports certification by similarity. If the device uses a Thread Certified Component (from the chosen silicon supplier) and makes no firmware changes, certification can be granted through a paperwork application without using an ATL.

The last step is to set up a production flow for obtaining and injecting a Device Attestation Certificate (DAC) into every device. NXP's EdgeLock 2GO service is a CSA approved Matter Product Attestation Authority which allows NXP to deliver Matter DACs to customers' manufacturing sites over the cloud.

The product can then be launched to the market and be a part of delivering the autonomous home experience to consumers! And, enhancements and new security vulnerability patches can be seamlessly delivered to users through over-the-air updates.

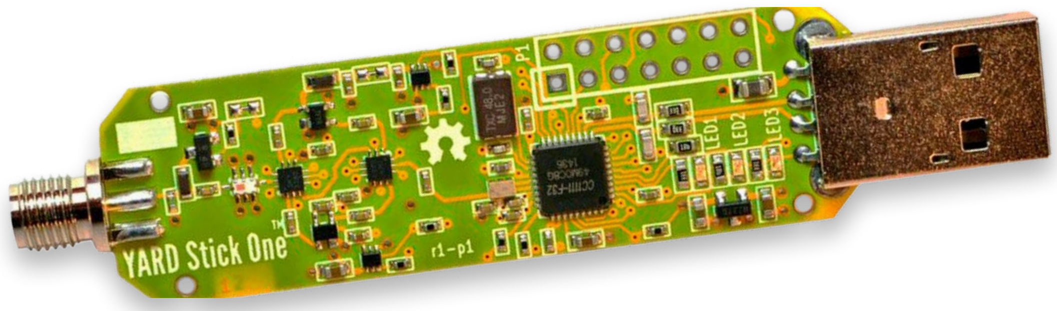
230488-01

WEB LINKS

[1] Matter – Making Smart Homes Smarter: https://www.nxp.com/webapp/sps/download/preDownload.jsp
[2] https://www.nxp.com/webapp/Download?colCode=MATTERSMRTHOMEWP
[3] Matter – NXP Semiconductors: http://www.nxp.com/matter

YARD Stick One

A Sub-1 GHz Wireless Test Tool



By Wim Ton (Ireland)

The YARD Stick One is a compact "hardware-defined radio" that can send and receive in the UHF band. It's a sort of breakout board with a USB interface to enable its use on hosts such as PCs and Raspberry Pis. The YARD Stick One is delivered with USB software preloaded in its 8051 core. The radio is controlled by writing a few dozen configuration registers, but the Python middleware can abstract many of the details.

The main advantage of the YARD Stick One from Great Scott Gadgets is that it's one of the cheaper devices (compared to the HackRF [1] or LimeSDR [2]) that can also transmit and works more or less "plug and play," whereas common, low-cost devices such as the RTL SDR dongles can only receive.

As all low-level functions are implemented by a CC1111 chip [3], using the radio is a matter of writing the configuration registers correctly. The CC1111 is targeted at complex Layer 2 protocols, with features such as sync words, framing, interleave and scrambling. As the CC1111 is designed as a SoC for commercial RF applications, the use of the YARD Stick One for signal analysis is very limited. Unless the system under test uses a similar SoC, it's less frustrating and cheaper to use an SDR.

The YARD Stick One is only supported by *rfcat*, Python-based middleware that abstracts the most-used options in a sort of descriptive methods. For fine-tuning, *rfcat* also offers raw register access.

As delivered, the device is a bare board, and must be treated with suitable care. Third-party enclosures are available.

The term "sub-1 GHz" is a bit broad; the YARD Stick One is limited by its TI CC1111 radio, which covers the lower UHF ISM bands: 300–928 MHz. Notably, the 13.56 MHz band, used for RFID, is not covered.

The YARD Stick One works slightly more stably on Linux than on Windows 10. Windows often does not recognize the device at all, but you can get away with repeatedly unplugging and reinserting the device on Linux.

Installing the Software

Using the YARD Stick One requires a good understanding of OSI layers 1 and 2. Some Python knowledge and familiarity with the intended OS is also beneficial to iron out installation issues. The software recommended in the Elektor store [4] was installed on Windows 10, Kali, and Ubuntu 18.

Installation of rfcat [5] on Linux with Python 3.10 worked well. The only difference from the documentation was that rfcat must be started with

```
./rfcat
```

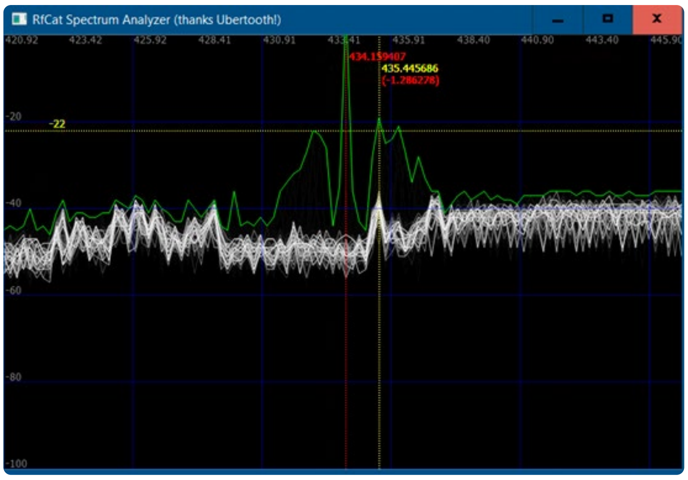


Figure 1: Rfcat spectrum display.

If you receive an `Error in resetup()`, unplug and reinsert the device.

Windows 10: Do not install Python via Microsoft Store, as it messes up the file permissions; install manually for all users. Additionally, you should enter:

```
pip install Cython
```

In any case, install with admin privileges (see where the Linux instructions require `sudo`). And have a VC > 14 installed.

Required tweak if you get an error message about "collections not callable:" add `.abc` in *C:\Program Files\Python310\Lib\site-packages\pyreadline\py3k_compat.py* on line 8:

```
return isinstance(x, collections.abc.Callable)
```

The *pyreadline* package is said to be only required for Windows.

Install the *libusb-win32* driver. The easiest way is probably using *Zadig* [6], which usually comes with *SDR#*. If the device is absent, you get the "No Dongle Found" exception from *rfcat*. In case of a "ChipconUsbTimeoutException," unplug and reinsert the device. All in all, installation and use on Windows 10 is a bit more tricky than on Linux. It is mandatory to call setmodeIDLE() at the end of the script, or a "device not found" error will follow on the next start.

The controller used in the CC1111 is a MCS51 variant and needs SDCC (Small Device C Compiler) [7] version 3.5 or lower. It requires some manual work during installation, as the current version is 4.x. However, many users will just be using the *rfcat* firmware.

Using the YARD Stick One

The CC1111 radio does all the low-level work, adding and removing pre- and post-ambles, sync words, CRC, plus modulation and demodulation. The radio must be fully configured before use, as the reset configuration is useless. Writing a small Python program as outlined in [8] saves a lot of typing and errors.

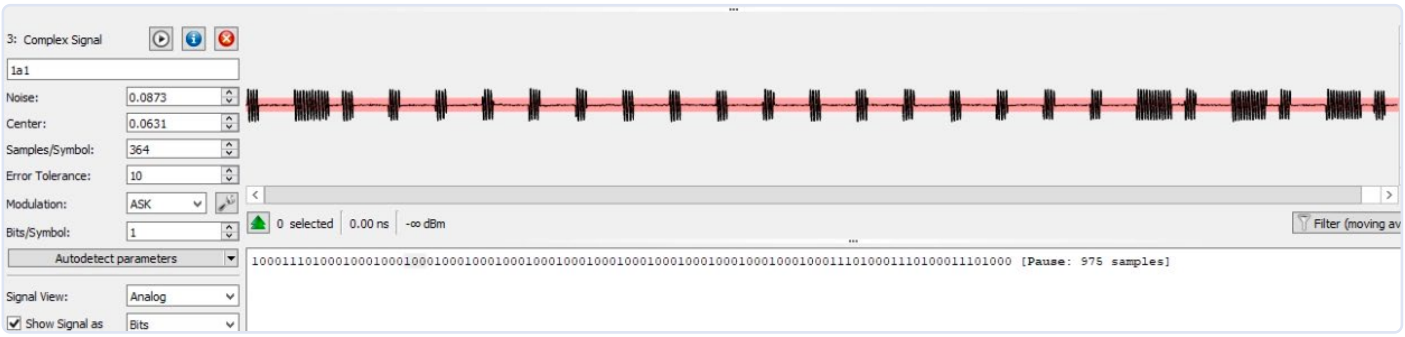
Rfcat comes with a spectrum display as well. Calling it an "analyzer" is a bit of a stretch — most of the simpler devices have a limited bandwidth and dynamic range, unlike the boat anchors such as the HP141 or HP181 that can display 1 GHz bandwidth at an 80 dB dynamic range.

To use the YARD Stick One as a receiver, the Layer 1 and Layer 2 properties must be configured correctly, or the radio will ignore the packet. To analyze an unknown signal, an additional SDR is needed; the cheapest hardware for this are the RTL dongles. Besides using GNU Radio and Audacity as shown in [9], the Universal Radio Hacker [10] provides a more integrated workflow for signal analysis and replay. Alternatively, the setting of the peer's radio chip can be sniffed from the hardware interface if the type of chip is known, as shown in [11].

Using the YARD Stick One as a generic receiver is rather awkward: when using too generic settings a lot of noise is received; with too tight settings everything is filtered out. It might work with a variable attenuator at the input, but I did not have one at hand.

To help with the configuration of the many registers of the CC1111, the utility SmartRF Studio from TI [12] helps. The calculated values can be written to the YARD Stick One with the appropriate `setXxx(value)` function.

The firmware as delivered with the YARD Stick One in the Elektor Store acts as a bridge between the CC1111 registers and the USB



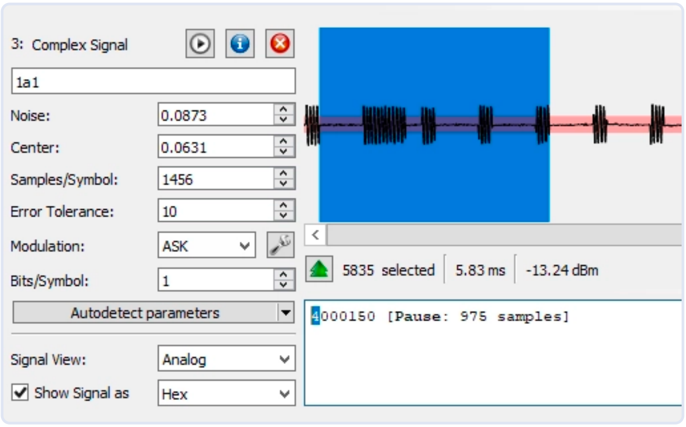
1a1

Noise: 0.0873
Center: 0.0631
Samples/Symbol: 364
Error Tolerance: 10
Modulation: ASK
Bits/Symbol: 1
Signal View: Analog
Show Signal as: Bits

0 selected | 0.00 ns | -∞ dBm

```
1000111010001000100010001000100010001000100010001000100010001000100010001000111010001110100011101000 [Pause: 975 samples]
```

Figure 2: Symbol view.


3: Complex Signal

1a1

Noise: 0.0873

Center: 0.0631

Samples/Symbol: 1456

Error Tolerance: 10

Modulation: ASK

Bits/Symbol: 1

Autodetect parameters

Signal View: Analog

Show Signal as Hex

5835 selected | 5.83 ms | -13.24 dBm

```
4000150 [Pause: 975 samples]
```

Figure 3: Data view.

interface. After a Python exception, the YARD Stick One must be unplugged and reinserted; otherwise, *rflib* does not find it anymore. The following example shows a PWM signal with ASK modulation, which is very common for simple remotes. The screenshots are from URH.

Each data bit consists of four symbols, a 0 is transmitted as 1000 and a 1 is transmitted as 1110, so this must be sent to the YARD Stick One as `8e888888888888888888e8e8e8`. One symbol is 0.484 ms, so the baud rate must be set to 2744.

Documentation

The documentation [4] pointed to by the Elektor Store is very sparse. The forum indicated by Great Scott Gadgets is of limited use. The *rfcat* git repository gives a lot of information on building and downloading the YARD Stick One firmware.There are some tutorials on the Internet, see for example [13]. Please adapt to the local rules for the ISM band.

Verdict

The YARD Stick One is not cheap for what it offers and has a rather steep learning curve. The software isn't perfect, and the documentation is on the lighter side. For analysis only, a simple SDR receiver is a much better choice. For transmitting, you have a generic SDR transceiver for €150 more (e.g., HackRF One or Adalm Pluto).

The YARD Stick One might be of use when you concentrate on the specific protocols supported by this family of radio SoCs. (See IM-Me [14] as mentioned on the Elektor site.) For dedicated applications, one could consider a CC111x breakout board [15] from China connected to an Arduino, which avoids the awkward USB communication.

230388-01

Questions or Comments?

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

WEB LINKS

[1] Denis Meyer, "HackRF One SDR Transceiver," elektormagazine.com:
https://elektormagazine.com/news/hack-rf-one-sdr-transceiver

[2] Jan Buiting, "LimeSDR Mini," elektormagazine.com:
https://elektormagazine.com/news/digital-tv-transmitter-based-on-raspberry-pi-zero-and-limesdr-mini

[3] CC1110Fx / CC1111Fx datasheet: https://ti.com/lit/gpn/cc1110-cc1111

[4] YARD Stick Software & Documentation: https://github.com/greatscottgadgets/yardstick

[5] RfCat GitHub: https://github.com/atlas0fd00m/rfcat

[6] Zadig: https://zadig.akeo.ie/

[7] SDCC: http://sdcc.sourceforge.net/

[8] YARD Stick One notes: https://bit.ly/3ql6mjo

[9] Hacking Everything with RF and Software-Defined Radio - Part 1: https://bit.ly/3qpIGul

[10] Universal Radio Hacker: https://github.com/jopohl/urh

[11] Radio Communication Analysis using RfCat: https://bit.ly/42kxvAj

[12] SmartRF Studio from TI: http://ti.com/tool/smartrftm-studio

[13] Hacking Everything with RF and Software Defined Radio - Part 2: https://bit.ly/43ITm5D

[14] IM-me: http://ossmann.blogspot.com/2010/03/16-pocket-spectrum-analyzer.html

[15] CC111x breakout board: https://aliexpress.com/item/32963409008.html

Latching Relays

Peculiar Parts, the Series

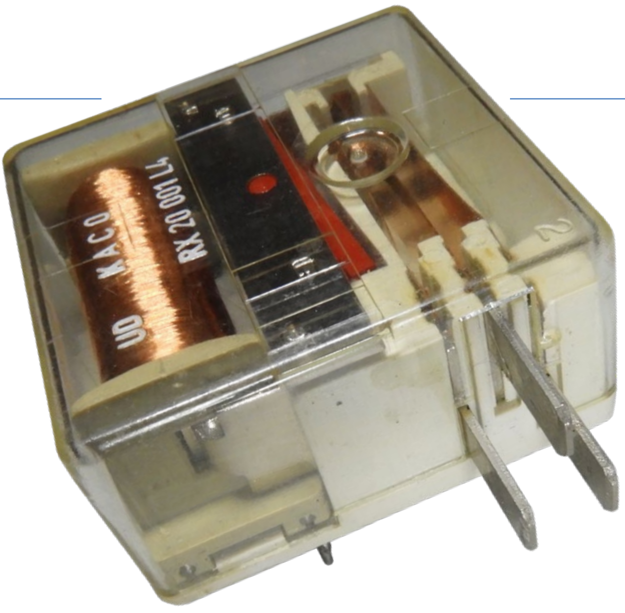


Figure 1: Typical power latching relay used in remote mains power switching devices.

By David Ashton (Australia)

It's easy to make a standard relay latch. Use a spare contact to switch power to the coil as soon as it is operated, and the relay will stay operated or "latched." But this still requires a current in the coil, and true latching relays stay latched with no coil current.

Latching relays can perform the trick either with two coils — one to operate the relay and one to release it — or they use one coil, where one polarity operates the relay and the reverse polarity releases it. With both types, the advantage is that only a short pulse of coil current is sufficient to change the relay's state, and thereafter the relay remains in that state until you pulse the other coil or give it a pulse of reverse polarity. Some types depend on permanent magnets to accomplish the change of state.

I have come across various types of latching relays in frequency-injection devices. These are basically devices on the electricity network that switch power to streetlights or hot water systems and the like; they can be remotely controlled using a special coded frequency injected onto the power wires (hence the name). Because latching relays are used, these devices consume almost no power in either state, and require only a short pulse of power to change it.

Latching relays would also be useful in other applications where power needs to be switched without consuming much of it. For example, in solar charge controllers where you want the maximum power from a solar panel to go towards charging a battery, but also to be able to switch the solar panel out of circuit during the hours of darkness — or switch the load off when the battery is getting low, without constant draw on the battery.

Figure 1 shows a typical power latching relay used in remote mains power switching devices mentioned above. This one has a 24 V coil with PCB pins and 10 A contacts with spade connectors. A red projection on top shows which position it is in.

Figure 2 shows some smaller Panasonic relays with four changeover contacts. These are two-coil latching relays from Panasonic, type DS4E-ML2-DC5V, with 1500 V isolation. Smaller latching relays like this would be useful in battery-powered equipment for power or signal switching without constant battery consumption — maybe switching attenuators in and out, for example.

Latching relays are components that find a home in very specific designs, where their power-saving properties make them very useful. For simplicity, isolation, and low-power consumption, there is not much that can beat their specifications. It's worth knowing about them, as they can perform functions that would otherwise be very difficult to obtain.

220653-01

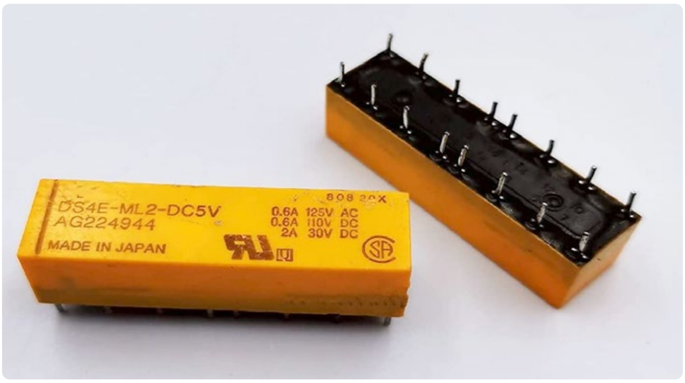
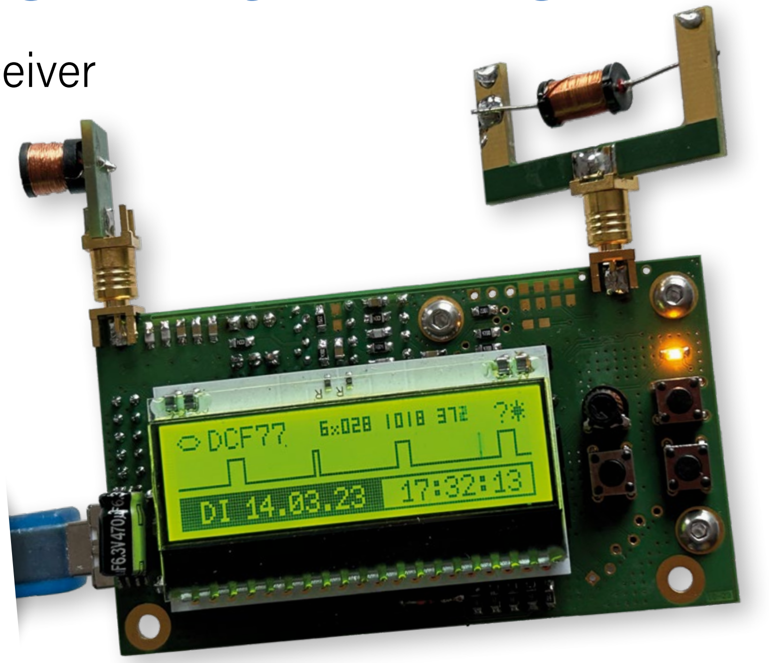


Figure 2: 2-coil latching relays from Panasonic with 4 changeover contacts.

PIC O'Clock
In Touch with Time

Designing an SDR Time Signal Receiver



By Marco Rohleder (Germany)

When you're designing a time signal receiver, there's no getting around the concept of software-defined radio (SDR). Elektor has published many articles about software for decoding these signals, and here we focus on the required hardware. In this connection, the aim was to manage with a minimum of processing power by using a simple 8-bit microcontroller equipped with peripheral functions that work independently of the CPU. Remarkably enough, at the end of the development process described here, the PIC MCU used was even able to receive and decode two different time signals at the same time.

How can one construct a time signal receiver using readily available resources? The key component required for extracting the time signal is a microcontroller (MCU), so an obvious approach is to utilize the existing resources within the microcontroller as much as possible. This means that the receiver should rely heavily on software and minimize the use of additional hardware.

Moreover, the objective is to avoid using exotic and hard-to-find components, instead favoring standard components whenever possible. This is especially true for SMD components. Capacitors and resistors in 0805 packages can be easily soldered by hand, saving space on the circuit board. However, soldering a four-lead IC with very small lead spacing is more challenging.

Along with the DCF77 signal at 77.5 kHz from Mainflingen near Frankfurt, there are many other transmitters in Europe and elsewhere in the world that broadcast the current time, including MSF60 at 60 kHz in Great Britain, TDF162 at 162 kHz in France, and WWVB at 60 kHz in the USA. In addition to dedicated time signal transmitters, there are also ripple control transmitters, such as DCF49 at 129.1 kHz (also in Mainflingen) and DCF39 at 139 kHz in Burg near Magdeburg, that transmit the current time of day. As a bonus, DCF39 and DCF49 have a transmit power of 100 kW, significantly more than the 50 kW transmit power of DCF77. Various meteorological services, such as maritime weather reports, are also still active on these frequencies. Wouldn't it be nice to be able to receive all these stations (and even more) in the longwave band as desired?

Getting Started Is Always Difficult

My first attempts to amplify the DCF77 signal from a ferrite rod antenna tuned to 77.5 kHz and view the carrier frequency on an oscilloscope to see how its amplitude varied at a 1 Hz rate were bitterly disappointing. Unfortunately, there is a lot of noise present in the longwave band. Switching power supplies and LED lighting systems in all shapes and sizes, mainly of Asian origin, create a noise level that in some cases is considerably higher than the desired signal. When you look at the mishmash of all possible signals, it's difficult to believe that the DCF77 signal can be extracted from this.

The commercial modules are apparently designed as simple straight-through receivers using a 77.5 kHz crystal as a very narrow bandpass filter. Although this approach is simple and low-cost, it means you can only receive signals at 77.5 kHz ±10 Hz. Ultimately, it became clear that without massive (and as it later turned out, digital) filtration, the goal of trouble-free time signal reception was doomed to remain elusive.

First, let's look at the block diagram of the fully configured time signal receiver (**Figure 1**).

We'll go through these blocks one by one and fill them with content.

Preselector with MOSFET Variable Capacitance

Even the most effective and steep-edged filters implemented in software and the best and fastest A/D converters are of no use if they are swamped by interference and the desired signal is lost in the noise. There's no avoiding a preselector that eliminates the bulk of the noise.

An LC resonant circuit with a variable capacitor (**Figure 2**) is often used as a preselector. The resonant frequency of the tuned circuit is given by the formula:

$$f_0 = \frac{1}{2\pi\sqrt{LC}} \qquad \textbf{Formel 1}$$

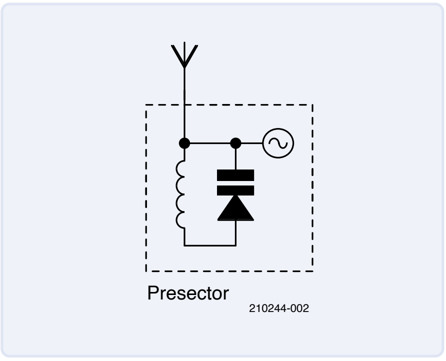


Figure 2: Basic preselector circuit.

If you use a ferrite rod antenna with a typical inductance of 1500 µH, as commonly used for reception in the longwave band, you will need a variable capacitor with a range of 400 pF to 6.7 nF if you want to cover the frequency range of 50 to 200 kHz. Unfortunately, there are no variable capacitance diodes available that meet this specification.

I recalled that every semiconductor device has an inherent (and unintentional) junction capacitance that generally depends on the size of the junction and is voltage-dependent. My first measurements on a power MOSFET looked promising, and, from a variety of different types, an IRF640 yielded the best results, with a capacitance of 500 pF at 20 V_{UDS} and 3 nF at 1 V_{UDS}. To make it easier to tune through the frequency range, I added a number of additional capacitors (C30...C34) that could be connected in parallel with the power MOSFETs (T35 and T36) as needed with the aid of bipolar small-signal transistors (T30...T34). With this arrangement, the preselector in **Figure 3** can be tuned over the range of around 30 to 300 kHz with an inductance of 1500 µH.

To allow the tuning range of the MOSFETs to be used as well as possible, they must be driven by a higher voltage than can be provided by the DAC in the microcontroller. Accordingly, an LM358 opamp is used to boost the signal from around 0...4.2 V to the required voltage range of 0...30 V.

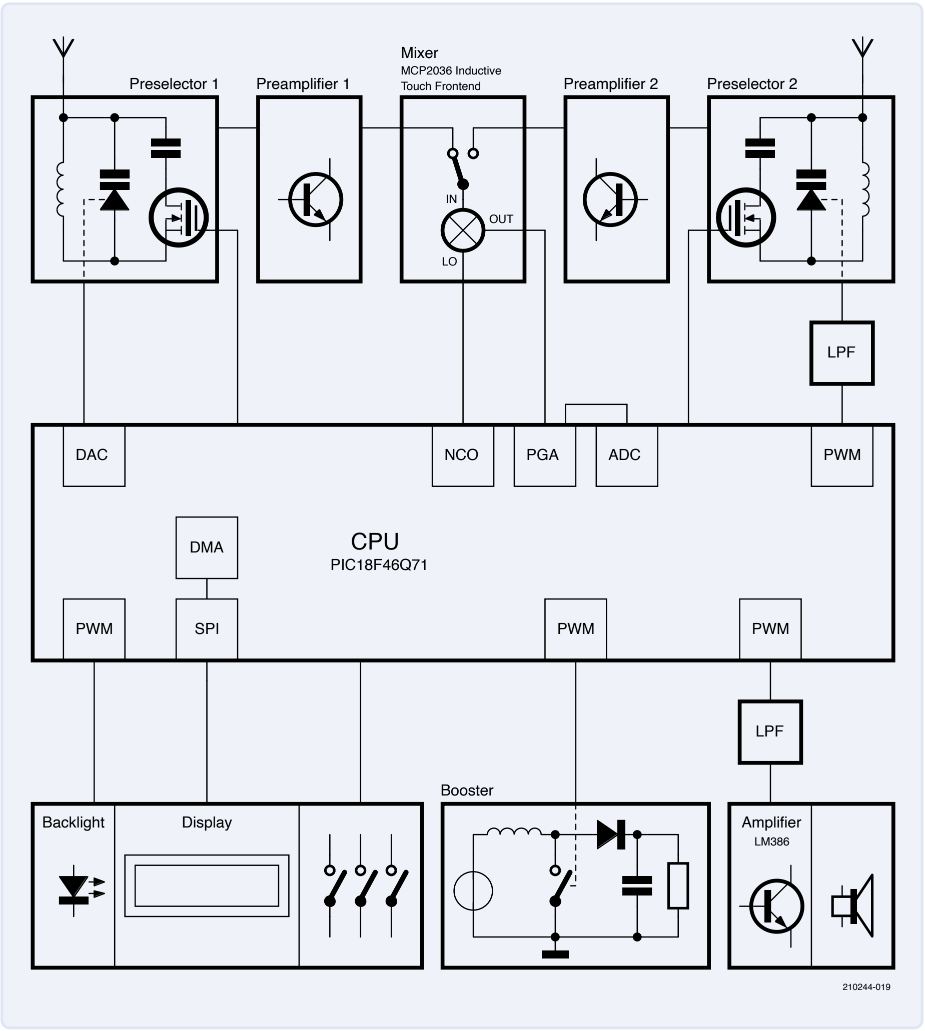


Figure 1: Time signal receiver block diagram.

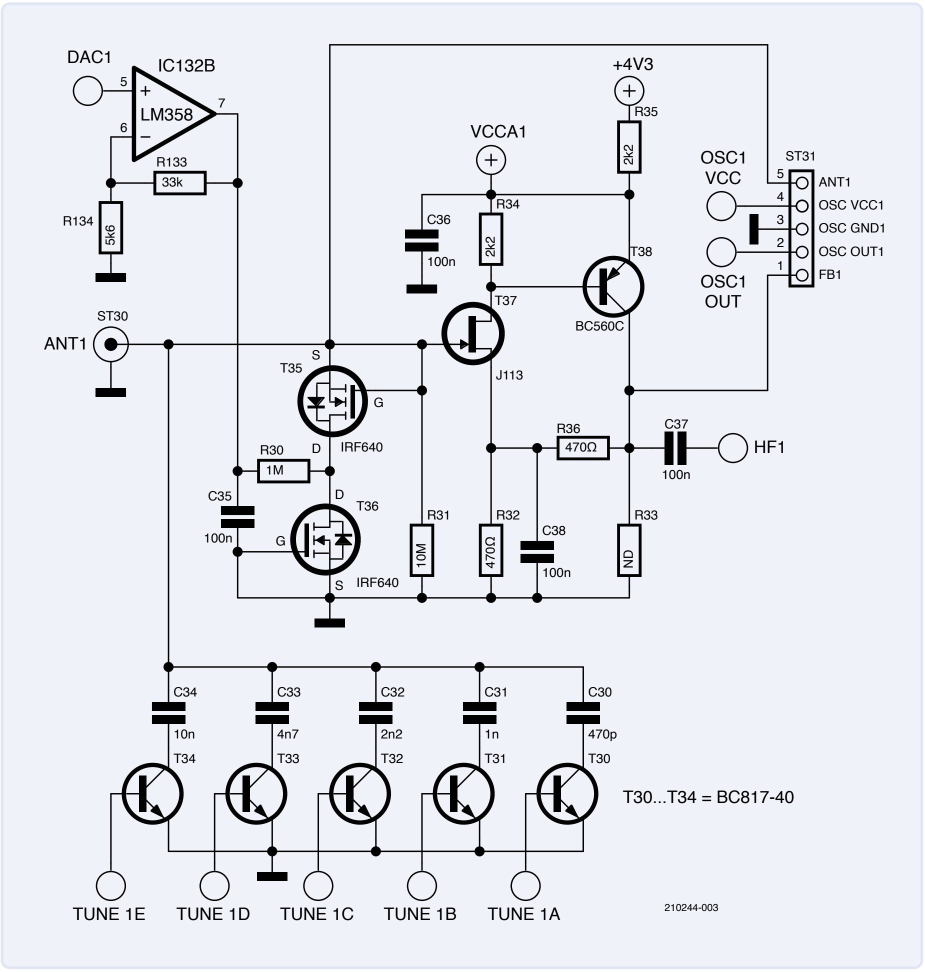


Figure 3: The preselector with the connected amplifier for the antenna signal.

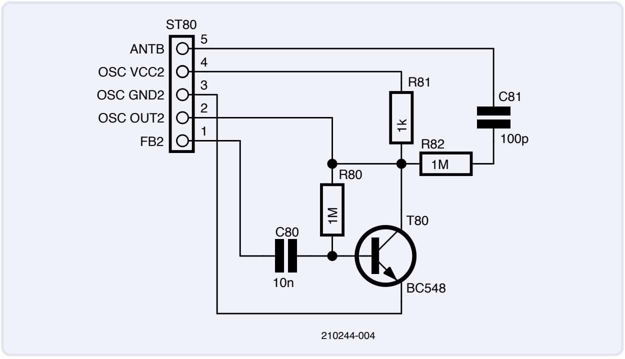


Figure 4: Oscillator implemented with transistor feedback.

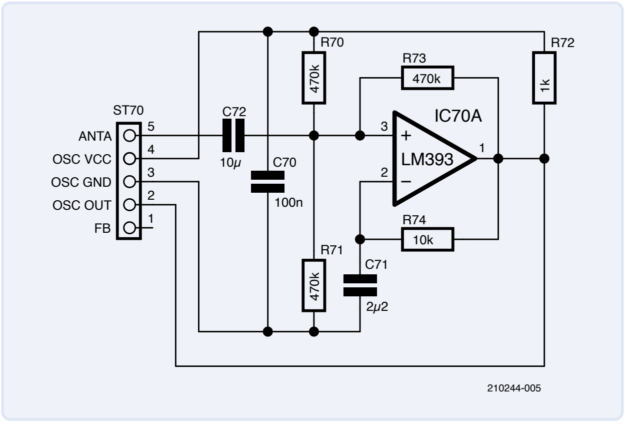


Figure 5: An oscillator based on an opamp.

Alternative Resonant Circuits

To allow different ideas to be tested, I used a connector (ST31, Figure 3) so that this part of the circuit could be swapped out.

Figure 4 shows a simple way to implement the feedback. Here, the required phase shift and additional gain are implemented using an extra transistor (T80). This circuit has the advantage that it only has a slight damping effect on the resonant circuit, but it turned out to be difficult to achieve reliable resonance over a wide frequency range.

The feedback turns the preselector into a resonant circuit whose oscillation frequency is approximately the same as the resonant frequency, but in the longwave band. There's a small difficulty: At the envisaged receive frequency, the phase shift from the input to the feedback changes, unfortunately, so the circuit does not always resonate at the expected resonant frequency.

On the second try, I adapted an old circuit that implements an LC meter with the aid of a comparator (**Figure 5**). It resonates over a wide range of inductance and capacitance values and matches the expected resonant frequency very well. As so often happens in electronics, however, an improvement in one place comes at the expense of a degradation somewhere else. Although the terminals and feedback were implemented with very high impedance, the input impedance was significantly higher than with the previously described solution. Three 470 kΩ resistors (R70, R71, and R73) connected in parallel in terms of HF, amounting to around 150 kΩ, strongly dampen the resonant circuit. In practice, this had hardly any effect on reception quality, other than that the overall gain had to be increased a bit.

For this variant, I would have liked to use one of the comparators integrated into the MCU. Although the pin multiplexer in the MCU offers a lot of different ways to feed out signals to different I/O pins, with intensive use of the analog peripherals you run into limits here, at least if you want to avoid routing the sensitive antenna signal crisscross over the board layout. For this reason, I opted for an external comparator (IC70) to keep the signal paths as short as possible.

Ultimately, I ended up with the circuit in **Figure 6** as a compromise. Here, the resonant circuit is reliably excited into resonance by a short pulse (200 ns) on ST31/90 (Pin 2) via R90/D90. Without feedback, however, the resonance only lasts for a few periods, which is too short to allow it to be measured accurately enough by the integrated timers. For this reason, the amplified signal is fed via ST31 (Pin 4) to one of the comparators integrated in the MCU, and the comparator output on ST31 (Pin 2) provides additional energy to keep the circuit resonating (see also Figure 2). Here the reference voltage on the comparator, which can be set via a DAC, determines how well you can match the expected resonance frequency. Maybe one of the readers of this article has an idea for how this can be adjusted automatically in normal operation so that it works reliably at different target frequencies.

No matter which of these three options you choose, in the end you have a digital signal of decent amplitude that you can feed to one of the counters in the microcontroller. As the desired frequency resolution does not need to be better than 100 Hz, it is sufficient to measure the frequency with a time window of 10 ms (1/100 s).

On startup, the software first measures the frequency ranges that can be achieved by connecting capacitors C30 to C34, in each case with the minimum and maximum tuning voltage. The best-match band is then selected for the definitive tuning process and the tuning voltage is adjusted stepwise upwards or downwards, starting at half the voltage range. The total time required to tune to a defined receive frequency is less than 100 ms.

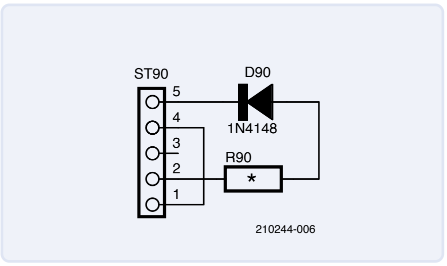


Figure 6: The oscillator is excited by a resistor and a diode.

Alternatively, you can excite the resonant circuit at a known frequency and adjust the capacitance until you get the least damping and the highest amplitude. For this, you can connect a very high-value resistor to the binding posts (ST31, Pin 2, Pin 5) and use a highly precise reference frequency on the I/O pin (more about this later) to excite the resonant circuit.

In addition, I placed a trace around the entire board layout that is also sufficient to excite the antenna at the desired tuning frequency. This works very well, but in comparison to the previously described solution, it takes a long time because many iterations are necessary. It therefore seems to me that the most reasonable approach is to combine the two solutions. First, you roughly approach the target frequency with the oscillator, and then do the fine-tuning by excitation and tuning to maximum resonance. The screenshot in **Figure 7** shows a typical calibration and tuning process at 77.5 kHz.

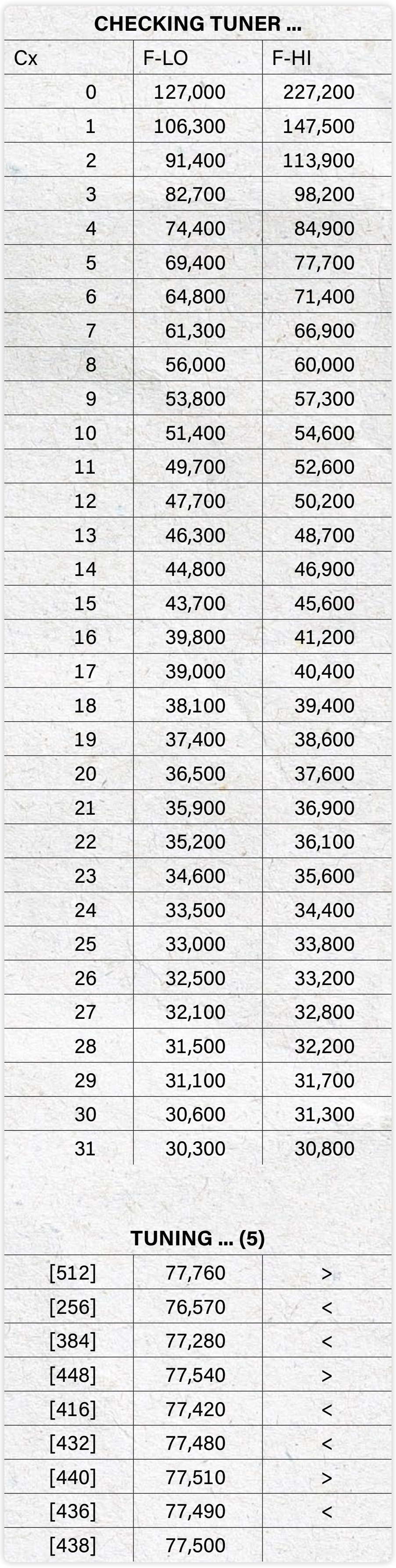
Unfortunately, as can be seen in **Figure 8**, the capacitance versus voltage characteristic of the junction capacitance is not linear (note the logarithmic scales on the chart). In other words, the capacitance changes are large at low voltages and become smaller and smaller at higher voltages.

To remedy this, we can take advantage of a special feature of the DAC in the microcontroller: It can be supplied with a variable internal reference voltage. For low output voltages, we choose a reference voltage of 1.024 V, resulting in a correspondingly low voltage swing. For higher tuning voltages, the reference voltage can be increased in steps up to 4.096 V.

Now, the signal from the preselector is available for further processing. Unfortunately, the A/D converter integrated in the microcontroller has a maximum conversion rate of 300 kHz.

Mixer and More

As the sampling rate for a software implementation of the receiver should be at least twice the highest received frequency, and preferably four times, the received signal must first be down-mixed to a lower frequency so that it can be further processed by the microcontroller.



CHECKING TUNER ...

Cx	F-LO	F-HI
0	127,000	227,200
1	106,300	147,500
2	91,400	113,900
3	82,700	98,200
4	74,400	84,900
5	69,400	77,700
6	64,800	71,400
7	61,300	66,900
8	56,000	60,000
9	53,800	57,300
10	51,400	54,600
11	49,700	52,600
12	47,700	50,200
13	46,300	48,700
14	44,800	46,900
15	43,700	45,600
16	39,800	41,200
17	39,000	40,400
18	38,100	39,400
19	37,400	38,600
20	36,500	37,600
21	35,900	36,900
22	35,200	36,100
23	34,600	35,600
24	33,500	34,400
25	33,000	33,800
26	32,500	33,200
27	32,100	32,800
28	31,500	32,200
29	31,100	31,700
30	30,600	31,300
31	30,300	30,800

TUNING ... (5)

[512]	77,760	>
[256]	76,570	<
[384]	77,280	<
[448]	77,540	>
[416]	77,420	<
[432]	77,480	<
[440]	77,510	>
[436]	77,490	<
[438]	77,500	

Figure 7: Checking the tuner.

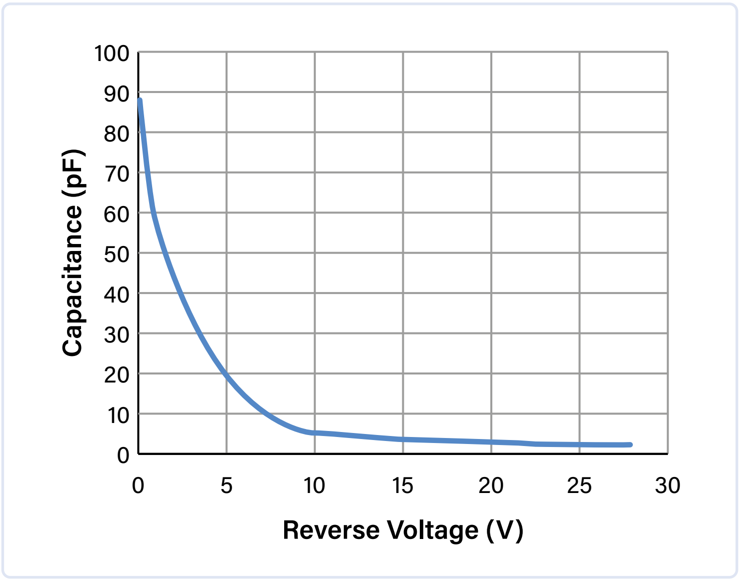


Figure 8: Junction capacitance nonlinear curve. (Source: Digikey — https://tinyurl.com/3js37yjn)

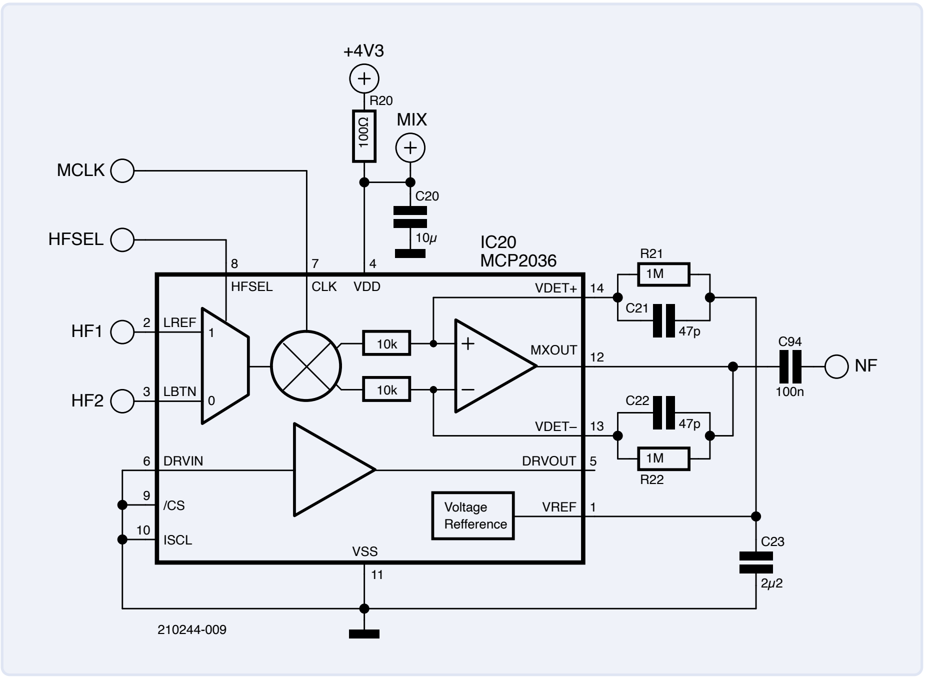


Figure 9: Internal structure of the MCP2036. (Source: Microchip)

More or less coincidentally, I learned about an interesting component, the MCP2036 from Microchip, which is actually intended to be used as a front-end for touch sensors, but also contains a mixer (**Figure 9**).

The signal to be mixed is fed in via the two inputs, LREF and LBTN. The local oscillator frequency is fed in via the CLK input as a digital signal, resulting in an intermediate frequency of several kilohertz at the output. This device also has other practical features, including a reference voltage source, a driver (not used here), and an opamp that can be used to further amplify the signal and, if necessary, low-pass filter the output to remove unwanted mixer products.

Then, the fully conditioned and down-mixed signal is fed to the A/D converter. There, the signal is sampled at four times the intermediate frequency to allow the I/Q components to be obtained downstream, enabling digital demodulation of the received signal.

But, to be able to receive amplitude-modulated stations such as the DCF77 time signal transmitter, the gain of the input stage must be regulated so that the input to the A/D converter is in the right range — not too high and not too low. For this, I used the opamp integrated into the PIC18F46Q71 microcontroller, configured as an inverting amplifier and connected in series with the input. A nice feature here is that a resistor network is also available to set the gain, and in this configuration it enables gain factors as high as 15. Like the DAC, the reference voltage of the ADC can be selected between 1.024 V and 4.096 V, representing an additional fourfold gain factor. And, if you use only eight of the A/D converter's available twelve bits, you get an additional adjustable 16-fold gain. Overall, this means that the gain can be adjusted over a range of 1:14,400 (14,400 = 15 × 15 × 16 × 4), as illustrated in **Figure 10**.

As six bits are actually enough to reliably decode the signal (as demonstrated below), the dynamic range can be increased to 1:57,600 (four times the above value) — more than 95 dB. That should be enough, no matter whether you're 2,000 km away from the transmitter on only a few meters away.

Now we just have to generate a variable local oscillator frequency for the mixer. For this as well, I used the microcontroller's peripherals. Most microcontrollers contain a timer/counter that can divide the clock signal by a factor *n*. This is sufficient for many applications, but the resolution is very poor. Suppose the reference frequency is 16 MHz. Using a divisor of 80 results in an output frequency of 200 kHz, but with the next larger divisor (81) the output frequency is 197.5 kHz — a step of 2.5 kHz.

Along with these standard timers, the PIC microcontroller used here also has an additional peripheral called a numeric controlled oscillator (NCO). As shown in **Figure 11**, this is an adder that each time adds a predefined value to its current result to generate a new result. This occurs at very short intervals, in this case at the microcontroller clock frequency of 64 MHz.

Each time the adder generates an overflow, either a result is generated or an output is toggled. At low frequencies, this allows resolutions as small as several tenths of a hertz. Of course, there is a certain amount of jitter in the output signal, but it can be ignored in this case.

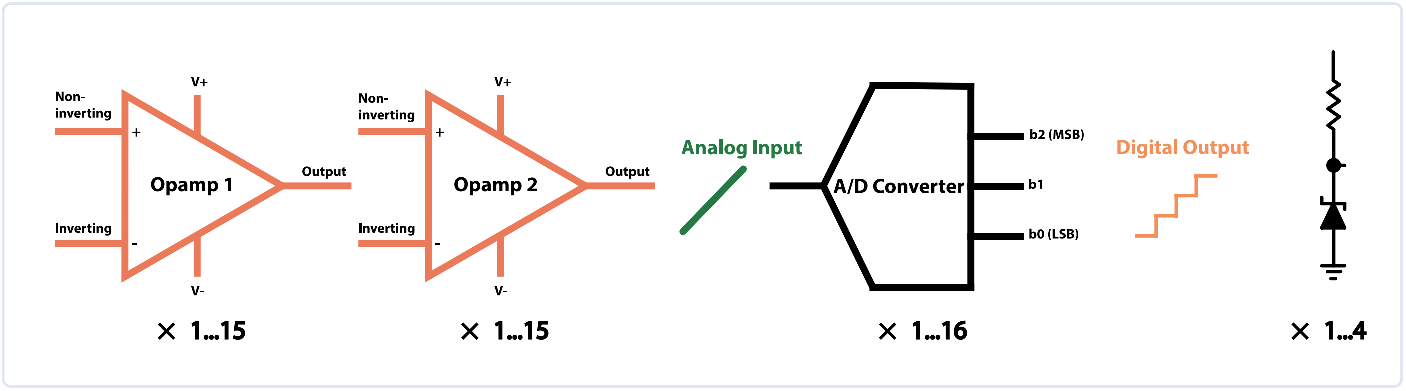


Figure 10: Adjustable overall gain.

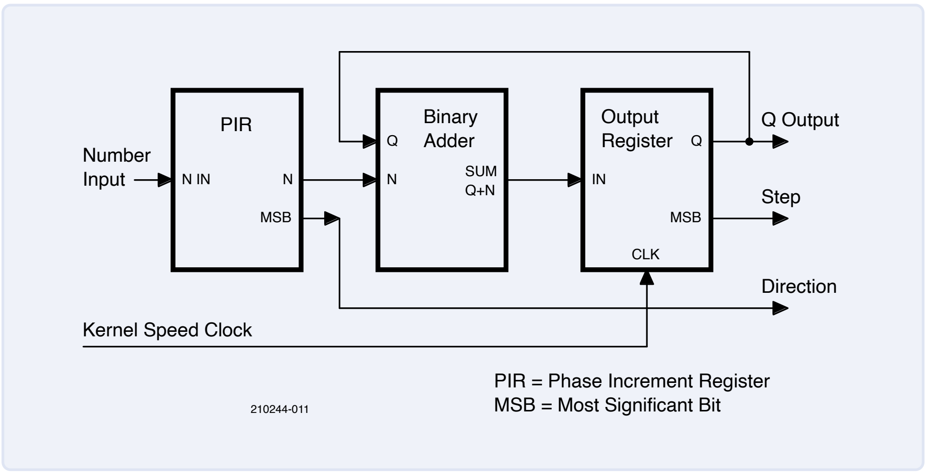


Figure 11: Numeric controlled oscillator.

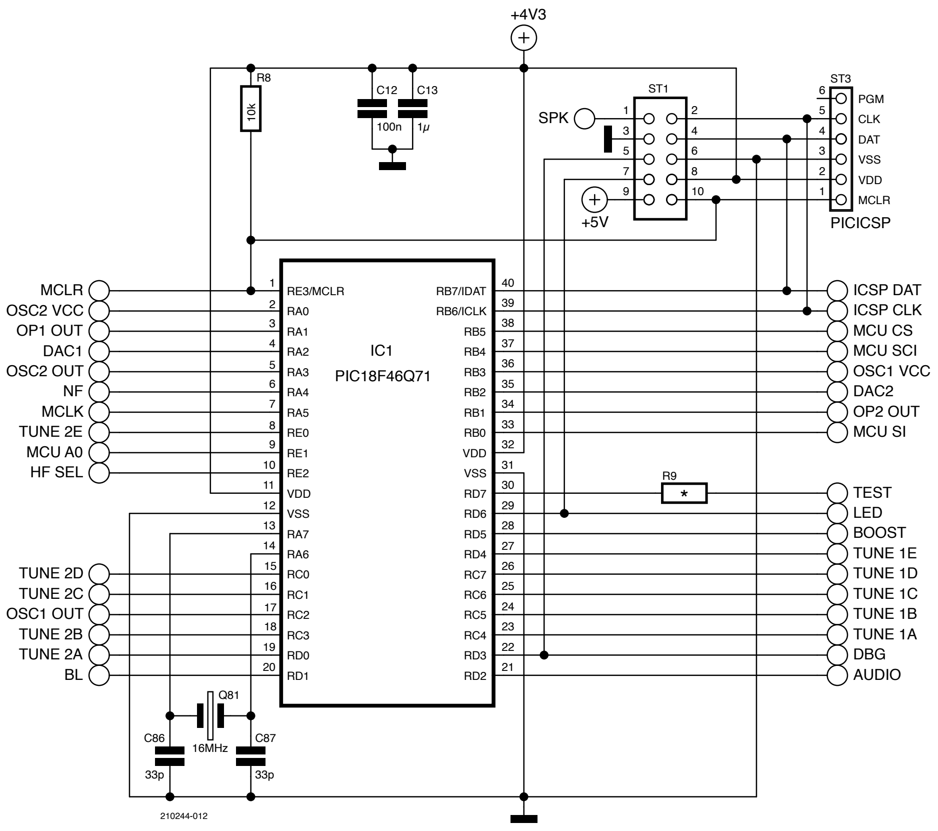


Figure 12: The brains of the time signal receiver: the PIC18F46.

The Microcontroller

I've already talked a lot about the microcontroller. With the many functions and tasks this MCU must provide and perform, it's very surprising that I opted for an advanced 8-bit microcontroller (Microchip PIC18F46Q71 [1]) instead of a high-performance 32-bit microcontroller. And what's more, the PIC18F46Q71 is available in a maker-friendly plastic DIP package. It has a wealth of peripheral hardware, which I used extensively here. As for extraordinary features in its class, it has a 12-bit ADC with a 300 kHz sample rate, two integrated opamps each with a gain-bandwidth product of 5.5 MHz, a multichannel DMA controller, multiple fast PWM channels with deadtime generator, two UARTs, and a numeric controlled oscillator (the heart of the receiver circuit). It also contains other interesting peripherals, including programmable logic cells (which, for example, can be used to generate a signal with a 90-degree phase shift), two fast comparators, and a hardware CRC unit, which are not used in this project.

The microcontroller is also equipped with a hardware multiplier, so it can provide the right processing power needed for complex computations, such as executing a Hilbert filter to suppress the image frequency.

Figure 12 shows the microcontroller with all of its input and output signals. ST3 is the ICSP programming port, and several I/Os are routed to connector ST1 for debugging purposes. This allows serial data to be transmitted using the UART, for example to display the operating range of the preselector or to configure additional high-resolution PWM signals to output analog values for debugging the individual stages of the SDR demodulator.

A Suitable GUI

Of course, the receiver should also have a user interface in the form of a small graphic display and control buttons, along with a

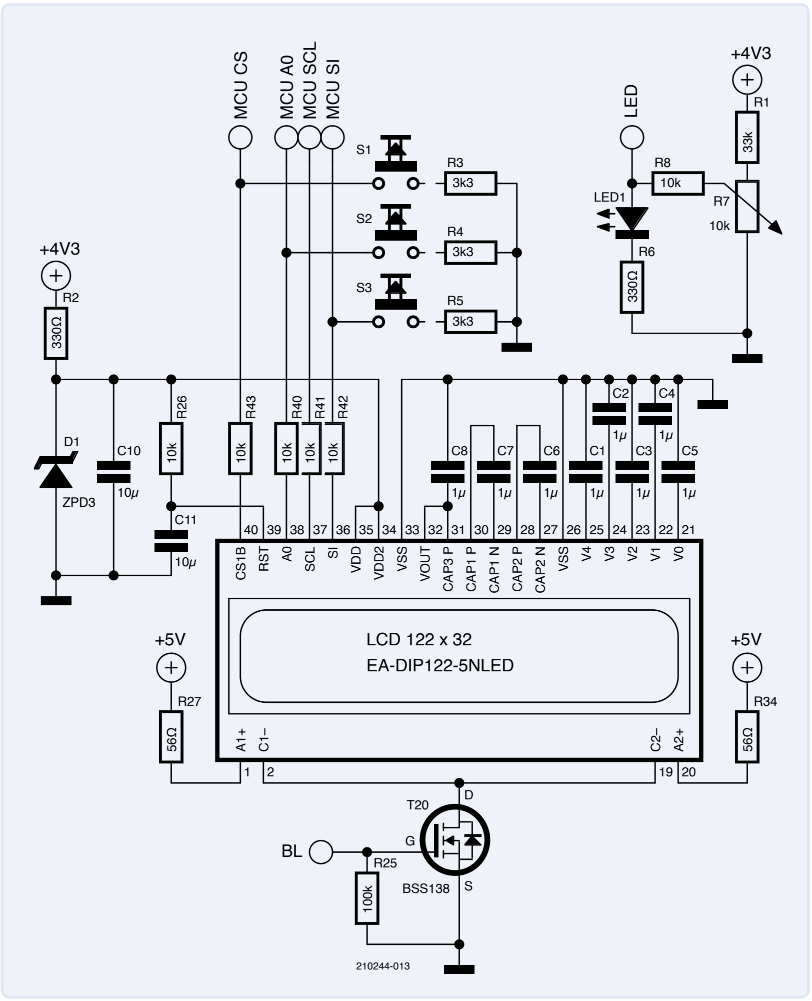


Figure 13: The user interface consists of a display, an LED, and some pushbuttons.

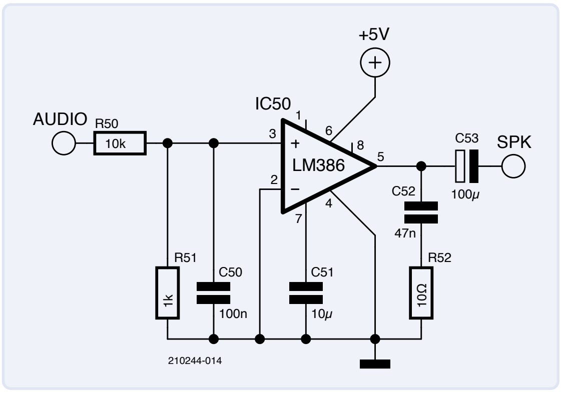


Figure 14: Audio output with the old faithful LM386.

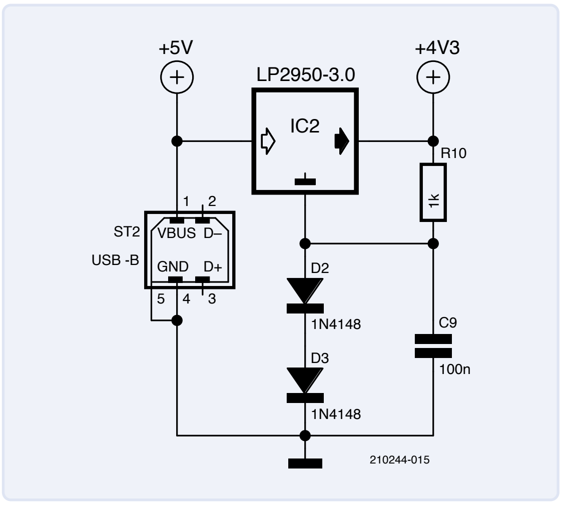


Figure 15: The 5 V supply voltage from the USB AC adapter is reduced to 4.4 V for the digital portion...

potentiometer (**Figure 13**). I opted for one of the well-known display-on-glass (DOG) modules, which is connected to the host microcontroller through an SPI interface. Annoyingly, the SPI interface of the display is implemented as one-way only, which means that data can be written to the display but cannot be read from the display.

For this reason, the graphic image to be displayed must first be created in the microcontroller RAM and then periodically sent to the display. For this, I again used one of the microcontroller's peripheral hardware units, in this case the DMA controller. It cyclically sends the content of the display RAM to the display via the SPI interface. This runs completely in the background and does not require any CPU processing power.

The overall current consumption of the display is only a few hundred microamperes, so I implemented the additional 3.3 V supply voltage with a Zener diode. For signal level conversion, I used series resistors in combination with parasitic bulk diodes present in the display inputs.

Audio
As some radio stations are also within the reception range (for example, BBC World at 198 kHz), I additionally included a loudspeaker and amplifier in the project. I aimed to make maximum use of the peripherals present in the microcontroller to implement the amplifier.

Initially, I opted for a Class-D design, with the output stage transistors driven digitally by the microcontroller and switched either fully on or fully off. Interference to the receiver could be avoided by choosing a suitable clock frequency with as few harmonics as possible in the receiver frequency range. But, what worked so well in the implementation of the voltage booster (see below) turned out not to be possible here, perhaps due to the significantly higher power or the considerably longer cable path to the loudspeakers, or both.

So, I built a conventional analog amplifier based on the tried-and-true LM386. The audio signal is first processed digitally by the microcontroller because it is present in digital form as I/Q components after the A/D conversion, in any case. At the output, the computed samples are fed to a PWM unit operating at a high clock rate, after which they pass through a low-pass filter and are then fed to the input of the amplifier IC (**Figure 14**).

The PWM unit used here can be clocked directly from the 64 MHz system clock, resulting in a PWM frequency of 15,625 kHz (64 MHz / 2^{12}) with a resolution of 12 bits. Along with the audio output, this also makes it possible to set the volume level. The chosen loudspeaker measures 40×40 mm and also serves as a mechanical brace.

Power Supply
A few brief words about the power supply (**Figure 15**): The idea was that the entire receiver should operate from a single USB port.

As the voltage on the USB port is largely unregulated, 3 V LDO voltage regulator IC2 first generates a clean operating voltage from the USB voltage for the digital part of the signal processing. The ground potential of the voltage regulator is raised by two silicon diodes, resulting in an output voltage (VDD) of 4.3 V. This is low enough to compensate for voltage variations at the USB port and high enough for the reference voltage sources used in the microcontroller.

Now, let's go back to the beginning: For the tuning voltage, the 5 V USB voltage must be raised to a much higher level. This is done by an unregulated boost converter with a refreshingly simple design (**Figure 16**). The PWM signal used to drive T70 is generated by the microcontroller with a suitable duty cycle. The Zener diode limits the output voltage to a maximum of 33 V. Although this circuit is not especially efficient, the current consumption is low and is essentially determined only by the opamp's quiescent current, so efficiency is not a significant issue in this case.

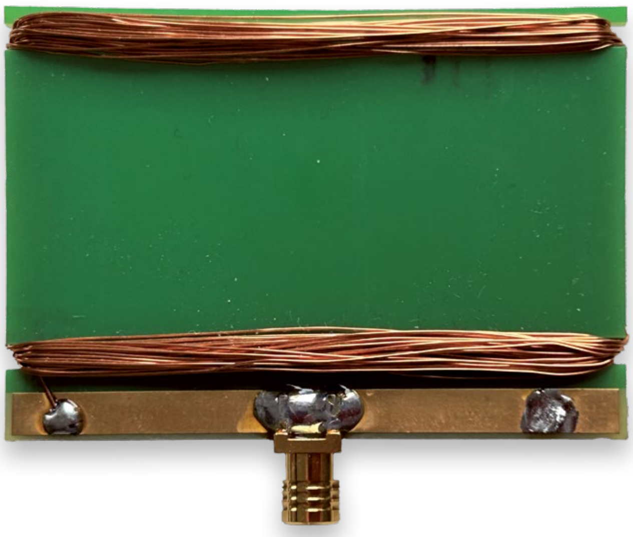


Figure 17: DIY loop antenna.

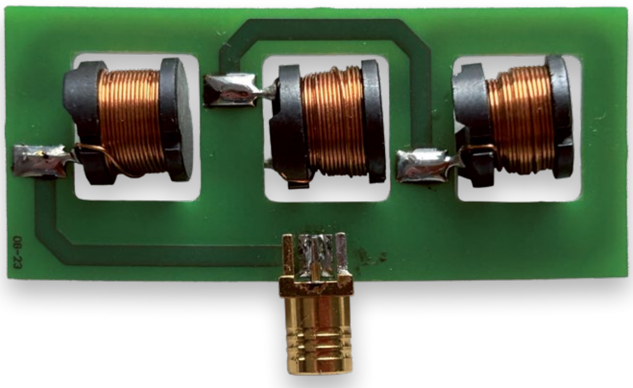


Figure 18: Good reception can be achieved using fixed inductors in an unusual configuration.

Antennas

The antenna is joined to the receiver through a matched pair of SMB connectors. This arrangement allows the antenna to be rotated, so it can be oriented to the transmitter direction as well as possible.

For the initial reception experiments, I removed a ferrite rod antenna from an old radio alarm clock and glued it to a piece of circuit board. Of course, I first had to remove the tuning capacitor because it is now simulated in the preselector.

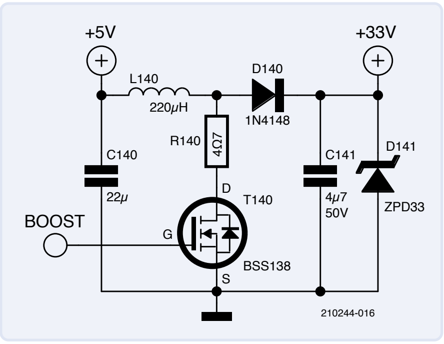


Figure 16: ...and boosted to 33 V for the tuning voltage.

As ready-made ferrite rod antennas and even bare ferrite rods have now become hard to find, and because winding coils is not everyone's favorite pastime, I went looking for an alternative antenna. On the clearance page of a mail-order electronics distributor, I found a loop antenna and glued it to a piece of circuit board material. The reception results were comparable to those with the ferrite rod antenna.

Based on that, I designed a circuit board with notches as a winding aid (**Figure 17**) and made the antenna myself. I had to use very thin enameled copper wire to make an antenna with the required number of turns. All in all, a very tedious process, but doable.

When digging around in my component drawers, my gaze fell upon a large bag of fixed inductors that I'd acquired a short time earlier. Would it be possible to use them to build an unconventional antenna? If several of them are connected in series, the arrangement is similar to a ferrite rod antenna, but divided into multiple parts (**Figure 18**). It turned out that, with this arrangement, I was able to achieve comparable reception results, with significantly lower signal amplitude but with astonishingly good reception quality: I could easily receive DCF77 over a distance of 800 km. I tried mounting the inductors both radially and axially, and they worked both ways. If you combine the two variants (alternating radial and axial orientation), you should have an antenna that's largely omnidirectional. If, for example, the field lines passing through the axially oriented inductors are at a maximum, the radially oriented inductors will not see any effective field lines, and vice versa.

Software

The software basically operates on the principle of a conventional SDR, which means the I/Q components are used for demodulation of the signal. How this works has already been described in many articles in Elektor [2] [3], so it's not necessary to go into all the detail here. The software excerpt in the **Two receivers are better than one** box is the only thing that is really new and therefore deserves mention.

This circuit makes use of many of the integrated peripherals, for example generating the local oscillator frequency for the mixer and using the DMA controller, so the load on

Two receivers are better than one
Although the circuit was originally only designed to receive data and time signal broadcasts in the longwave band, the mixer has a second switchable input, so I decided to add an additional preselector with its own antenna. This can be fitted with components for either longwave or medium-wave reception. As became apparent in the course of developing the software, this even allows signals at two different frequencies to be received at the same time.

Unfortunately, the microcontroller I used here has only one NCO, so I had to resort to a trick to generate the second local oscillator frequency. I used one of the two 16-bit universal timers to generate the lower local oscillator frequency. Although the timer is clocked by the 64 MHz system clock, it is possible to come fairly close to the desired local oscillator frequency. At 77.5 kHz (DCF77), for example, the error is only 200 Hz. If the second universal timer is used to generate the sampling rate for the intermediate frequency, it can be adjusted between 2,000 Hz and 2,200 Hz with a resolution of around 1 Hz, making it possible to compensate for the previously described error. The NCO of the other receiver circuit has to be adjusted according to the resulting new intermediate frequency.

Of course, the crystal used has only finite accuracy. If we assume the accuracy to be 50 ppm, the error at 77.5 kHz is 4 Hz. This is too much for the very narrowband filter used for decoding. Nevertheless, the error can be determined by software. If the signal from the desired transmitter can be received with an adequate signal level, the frequency correction can also be performed automatically. For this purpose, the software must measure over an interval of four periods whether the I/Q vector is rotating clockwise or counterclockwise. If this is done over a defined time period, it gives you an indication of how much the actual local oscillator frequency differs from the desired frequency. A higher-level control function can then reduce or increase the intermediate frequency to bring the difference close to zero. The following listing shows how this can be implemented in C.

```
#define FILTERDEPTH 25
typedef struct {
  int16_t Ihistory[FILTERDEPTH];
  int16_t Qhistory[FILTERDEPTH];
  int32_t Iaverage;
  int32_t Qaverage;
  uint8_t index;
  uint8_t lastQuadrant;
  int8_t phaseChange;
} IQ;
IQ *iq;
void IQcallback( int16_t I, int16_t Q ) {

  uint8_t quadrant =0;

  // moving average lowpass filter I
  iq->Iaverage -= iq->Ihistory[iq->index];
  iq->Ihistory[iq->index] = I;
  iq->Iaverage += I;

  // moving average lowpass filter Q
  iq->Qaverage -= iq->Qhistory[iq->index];
  iq->Qhistory[iq ->index] = Q;
  iq->Qaverage += Q;

  iq->index++;
  if( iq->index >= FILTERDPTH )  {
    iq->index =0;
  }
  // integrate phase shift > phase error >
     estimate frequency offset of crystal
  if( iq->Qaverage<0 ) quadrant =1;
  if( iq->Iaverage<0 ) quadrant ^= 3;
  if( iq->lastQuadrant == ( (quadrant+1) & 3 ) )
    iq->phaseChange--;
  if( iq->lastQuadrant == ( (quadrant-1) & 3 ) )
    iq->phaseChange++;
  if( iq->phaseChange < -90 )
    iq->phaseChange = 90;
  if( iq->phaseChange > 90 )
    iq->phaseChange = -90;
  iq->lastQuadrant = quadrant;
}
```

the microcontroller is very low. As a result, even this 8-bit microcontroller has enough processing power available to demodulate the received signal.

The last link in this chain is the GUI, which evaluates newly calculated values and updates the display as necessary.

Outlook
It's a familiar story: After building the first prototype, you come up with many new ideas about how to improve things on a second try. Nevertheless, I decided to stick with my first approach and implement it as planned. If I were designing this again, I would do the following:

- Add a rotary encoder to the user interface to make it easier to set the frequency.
- Try out the PIC18F26Q71 (the little brother of the PIC18F46Q71 I used), but use two of them. The required board space would be nearly the same, and this way I would have two NCOx, four opamps, and four comparators. From an HF perspective, this would allow better routing. A drawback would be the necessary communications between the two microcontrollers, but that could be kept within limits. One CPU could handle the digitization and processing of the received data, while the other CPU could support the GUI. As both microcontrollers would be operated from the same clock source, a one-wire UART interface operating in the Mb/s range would be easily possible.
- Replace the SMB connectors, which have become nearly unaffordable, by old-fashioned jack sockets. A 2.5 mm version would work well here, and rotating the antenna would still be possible. These plugs and jacks are also available

in three-way and four-way versions, so it would even be possible to relocate part of the circuit, such as the tuning capacitors, to the antenna assembly.

- Suitably modify the preselectors, so that the receiver would be able to receive signals in the medium-wave band (AM broadcast band) or allow it to be used for amateur radio purposes. Practical experiments have shown that the mixer can be operated up to the single-digit MHz range.

As you can see, SDR does not necessarily have to be implemented with high-speed 32-bit microcontrollers, which are often only available in packages that are hard to solder by hand. With clever use of the peripherals of an 8-bit microcontroller, it is possible to substitute for one or more functions and ultimately arrive at an easily implementable design.

As always, this sort of project lives or dies with the number of participants who contribute to the effort. In particular, implementing the software takes a lot of time. But perhaps someone out there will have the desire and enthusiasm to help move this project forward? Let us know!

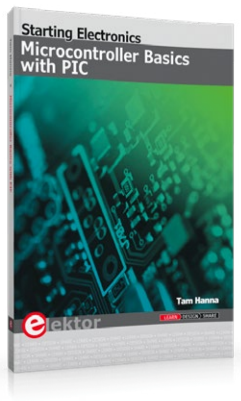
Translated by Kenneth Cox — 210244-01

Questions or Comments?

If you have any technical questions or comments about this article, feel free to contact the author at marco@happy-electrons.com or the Elektor editorial team at editor@elektor.com.

About the Author

Marco was born in 1970 and was interested in electronics from a very early age. He worked on electronics in his teens, studied electrical engineering at some point in time, and has been active in R&D for around 35 years. He is presently employed by a US semiconductor manufacturer. Marco is also a radio amateur and answers to the callsign DD9DD.



WEB LINKS

[1] MCU PIC18F46: https://microchip.com/en-us/product/PIC18F46Q71
[2] Martin Ossmann, "AVR Software Defined Radio," six-part series, Elektor 3/2012 to 10/2012: https://elektormagazine.com/magazine-archive/2012
[3] Martin Ossmann, "SDR Radio Controlled Clocks," Elektor 1/2023: https://elektormagazine.com/magazine/elektor-288/61435

PCBWay

YOUR ULTIMATE
PCB MANUFACTURING
PARTNER !

WHY CHOOSE PCBWAY?

www.pcbway.com
service@pcbway.com

TOP-QUALITY PCBS
Our cutting-edge technology ensures flawless PCBs for your projects.

LIGHTNING-FAST TURNAROUND
Experience industry-leading turnaround times.

HASSLE-FREE ORDERING
Simply upload your design, and we'll take care of the rest!

TOP-QUALITY PCBS
Our expert support team ensure a seamless experience from start to finish.

Source: Midjourney AI-generated image. Prompt: "/imagine prompt: Back of person with lots of papers under his arm moving toward factory --ar 3:5"

Due Diligence Directive

Business as Usual Will Not Do

By Priscilla Haring-Kuipers (The Netherlands)

Currently rolling through the EU is the Corporate Sustainability Due Diligence Directive or CSDDD [1]. The EU Parliament just voted generally to accept these new rules, and will continue to negotiate some of the finer points for another year. Things are about to change.

Our Sustainable Development Goals include the promotion of sustained, inclusive and sustainable economic growth. In the Paris Agreement, we promised to keep global warming within 1.5°C, and the private sector, with all their investments, must be in line with this if we are to keep our promises. Under the European Climate Law, we are committed to being climate-neutral by 2050 and to reduce our emissions by at least 55% in 2030. All of this means that there must be a change in the way companies produce and procure. Business as usual will not do.

Many issues in our global supply chains would be hard to tackle by any one company or even one country. These new due diligence rules intend to even the playing field within the EU market and prevent fragmentation in law. Germany already has a Supply Chain Act or "Lieferkettensorgfaltspflichtengesetz" [2], while other EU countries have limited requirements. By connecting all the big companies of many countries under common EU legislation, the power to effectively and sufficiently address environmental concerns and social inequality can be combined. Moving together is better.

The Ask

This directive concerns big EU companies with hundreds of employees and millions in turnover, as well as non-EU companies that make millions in the EU market. At its core, the CSDDD [3] is a way to force these big companies to take responsibility "by carrying out the following actions:

(a) integrating due diligence into their policies
(b) identifying actual or potential adverse impacts
(c) preventing and mitigating potential adverse impacts, and bringing actual adverse impacts to an end and minimizing their extent;
(d) establishing and maintaining a complaint procedure;
(e) monitoring the effectiveness of their due diligence policy and measures;
(f) publicly communicating on due diligence"

Companies will need to investigate themselves and their entire supply chain. When uncovering 'adverse impacts,' they must create corrective action plans with clear timelines and qualitative as well as quantitative metrics for improvement. Workers, unions and civil society organizations must be able to make human rights violations or negative environmental impacts known. Companies have to provide a complaint procedure to support this, and inform relevant parties, as well as provide appropriate followup. Companies need to check on the implementation and effectiveness of their policies and on those of their business partners continuously. Such check-ins must be done at least once a year, or when new issues arise in the supply chain. All of this must be done transparently and communicated publicly.

Sanctions are mentioned in the directive, but it is left to every EU country to enforce them in line with national laws and proportional to the company's revenue. In the EU, they will set up a European Network of Supervisory Authorities to help and oversee countries do this.

A director of such a big company has a duty to oversee the corporate due diligence, to establish a code of conduct and to integrate this into the corporate strategy. The CEO must ensure that sustainability, human rights, climate change, and environmental consequences are properly considered in the short, medium, and long run. Making a profit is not a valid counterargument.

Adjust to Size

Big companies will be liable if they fail in their due diligence and preventable bad things happen or continue to happen. This does not mean that companies have to guarantee that bad things will never happen. Companies must take measures appropriate to their level of power in the chain and respond to human and environmental rights violations when they find them (and they are obligated to look).

The CSDD protects human and environmental rights throughout the lifecycle of a product and throughout the entire chain. This should mean that big digital companies are going to have to include the manufacturing of their hardware as part of their responsibility.

Small and medium-sized enterprises (SMEs), which make up 99% of all companies in the EU, are not included in the CSDDD. However, big companies will have demands of any SME they work with in order to be able to fulfil their own due diligence duties. EU countries are expected to support their SMEs by setting up dedicated websites, portals, and platforms — and perhaps provide financial support to build capacity. Big companies are expected to invest in the SMEs with whom they work, in order to help them to comply.

More Moral

I think this directive is a good example of how law slowly becomes a formal version of the values we hold as a society. Of course, you don't have to wait around for laws to be established, and it makes good business and moral sense to be ahead of the curve. One example of this is RS Group, providers of industrial and electronics products to engineers. They have already started to build an ethical supply chain seriously [4].

Many SMEs will soon discover what power (if any) they have in their supply chain, and, perhaps for the first time, undertake an ethical examination of their companies' actions. This forced self-reflection will likely come with compulsory forms to fill, new reporting practices and self-certifying quality control marks, along with big companies usurping more parts of the supply chain in order to maintain control and fulfil their legal responsibilities. If you are part of an SME that works with bigger companies, yesterday would be a good time to start investigating and reporting on your due diligence. Many will already have made many moral business choices. Now, these need just be formalized and communicated.

230428-01

◄ *Source: Midjourney AI-generated image. Prompt: "/imagine prompt: EU Parliament with EU flags --ar 5:3"*

WEB LINKS

[1] EU CSDDD portal: https://bit.ly/43MAiUu
[2] Supply Chain Act (Wikipedia): https://en.wikipedia.org/wiki/Supply_Chain_Act
[3] Actual CSDDD: https://bit.ly/43Pg1Oa
[4] WEEF 2022 - Building an Ethical and Sustainable Supply Chain with Andrea Barrett: https://youtu.be/tuu88ePQwYQ

Starting Out in Electronics...

...Voltage Amplification

By Eric Bogers (Elektor)

So far, we have used the transistor as a switch and as a current amplifier. Of course, these are both very important in electronics, but it really gets fun when we can amplify voltages. That's precisely what we'll do in this installment.

But, before we jump into the emitter circuit, a word of caution is justified. This concerns the article in the May/June 2023 issue, and in particular the astable mutivibrator outlined in Figure 3 on page 40. As Elektor reader Ruedi Schwarzenbach from Switzerland wrote to us, this circuit has some potential dangers that can spoil the fun of novice electronics enthusiasts quite a bit.

What is actually the case: Although we have not mentioned a specific transistor type in the schematic and the text, it is in fact obvious to use the BC547, a general-purpose transistor, for this purpose. And, because the calculations in the text assume a supply voltage of 12 V, things could go wrong. This is because, according to the datasheet, the BC547 does not tolerate a high base/emitter reverse voltage, 5 to 6 V is just about the maximum. And that means that the transistors in the aforementioned Figure 3 circuit can already become damaged at a supply voltage of greater than 5 V.

Although we mentioned that "the base of the left-hand transistor carries a negative potential and is therefore turned off," one should bear in mind that at a negative potential of 5...6 V, the base/emitter path shows a kind of Zener effect. A current flows (the capacitor is in fact shorted via the two transistors), which could possibly cause the transistor to fail.

To avoid this kind of issue, it's better to use a voltage source of 4.5 V for the power supply (like a 4.5 V flat battery, for example) and to do all calculations with this voltage.

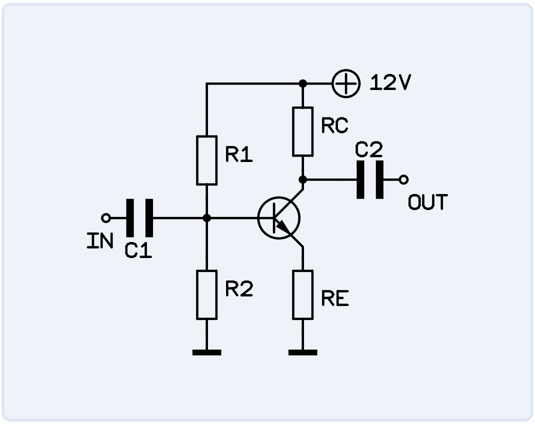


Figure 1: Common-emitter circuit.

The Emitter Circuit

The circuit shown in **Figure 1** involves an emitter circuit (not to be confused with an emitter-follower!), where the emitter is the common reference for the input and output signal. The emitter circuit is very similar to the collector circuit from the previous installment, except that a collector resistor is now present, while the output voltage is taken from the collector. The emitter resistor's main task now is to prevent the operating setpoint from drifting; its value is now a lot smaller. Assuming, again, a quiescent current of 1 mA, we end up with a value of 1 kΩ for the emitter resistor.

The output voltage can vary from 1 V to 12 V, i.e. within a range of 11 V; we choose the idle voltage in the middle of this — a voltage of 6.5 V is then present on the collector, while a voltage of 5.5 V falls across the collector resistor. We can now calculate RC:

$$R_C = \frac{U}{I} = \frac{5.5\ \text{V}}{1\ \text{mA}} = 5.5\ \text{k}\Omega$$

We take a default value of 5.6 kΩ for RC here. As for R2, we choose (arbitrarily) a value of 100 kΩ and calculate R1 as follows:

$$R1 = \frac{U_0 \cdot R2}{U_B} - R2 = \frac{12\ \text{V} \cdot 100\ \text{k}\Omega}{1.7\ \text{V}} - 100\ \text{k}\Omega = 605.9\ \text{k}\Omega$$

We take a default value of 560 kΩ for R1.

Just to be clear, that 1.7 V follows from the voltage across the emitter resistor of 1 kΩ at a quiescent current of 1 mA, plus the voltage of 0.7 V falling across the base/emitter junction.

The question now, of course, is where the voltage gain comes from and how large it is. If we assume that as a result of an applied AC voltage, the voltage at the base of the transistor increases by 0.1 V, the emitter voltage also increases by 0.1 V, which means that the current increases by 100 µA. As a result of this larger current, a higher voltage falls across the collector resistor (0.56 V more, to be precise). In other words, the voltage at the collector of the transistor is now 0.56 V lower.

A voltage *increase* at the base results in a corresponding voltage *decrease* at the collector, and not only that — the voltage change at the collector is greater than that at the base. The voltage change at the base is therefore amplified!

Also in this numerical example, we have neglected the difference between I_C and I_E. If we allow ourselves this modest freedom, we can write the following for the voltage gain of the common emitter circuit:

$$V_{OUT} = \frac{\Delta U_C}{\Delta U_B} \approx -\frac{R_C}{R_E}$$

So, in this example, the gain is -5.6×. The minus sign simply means that the input signal and the output signal are in antiphase. Or, put in another way, we are dealing here with an inverting amplifier.

It is possible to amplify a signal much more with this circuit, but in that case, a smaller emitter resistor and/or a higher supply voltage should be selected. Under all circumstances, we should avoid a 'design' like the one shown in **Figure 2**. For convenience, in that circuit, the designer omitted resistor R2 and emitter resistor RE.

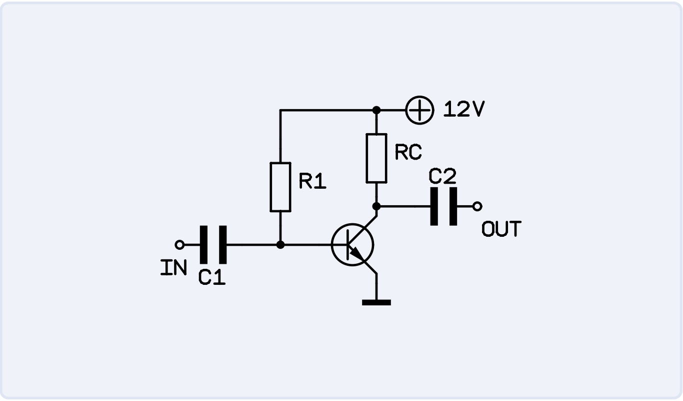


Figure 2: Catastrophic amplifier circuit — that's not how it should be done!

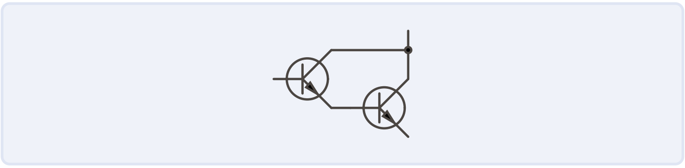


Figure 3: The Darlington circuit.

Because of the omitted emitter resistor, the circuit does deliver a very large gain, but the temperature stability of the overall circuit is awful: If the temperature changes by only a few degrees, the base current — and therefore also the collector current — increases (or decreases) and the operating setpoint drifts. Moreover, the current gain of a transistor is not a reliable parameter because it has a very strong exemplary spread — so the value of R1 can only be determined by experiment.

The omitted emitter resistor also results in a very low input impedance, so as a result the value of C1 should be selected to be much larger to avoid the lower cutoff frequency becoming too high.

To make a long story short: A circuit as shown in Figure 2 is not useful. Those who need very large amplification are better off connecting several amplifier stages in series or using an operational amplifier (opamp) beforehand.

The Darlington Circuit

For many applications, we need a larger current gain than can be provided by a single transistor. In such cases, the Darlington circuit (**Figure 3**) comes in handy. In this circuit, two transistors are switched in such a way that the emitter of the first transistor is directly connected to the base of the second transistor. Here, we may (as a good approximation) multiply the individual current gain factors by each other, so that values of 10,000 and more are achievable with small-signal transistors. It should be noted that the base/emitter voltages of the transistors are added together, which means that a Darlington circuit requires a setup voltage (bias) of at least about 1.4 V.

As for power transistors, whose current gain is often very modest, rather use the Darlington circuit. In many cases, both transistors in the circuit are already integrated in one package by the manufacturer — we then refer to it as a Darlington transistor.

The Constant-Current Source

If we use a LED for some occasion, then, in most cases, we already have a constant supply voltage available, and it is easy to calculate the required series resistance accordingly. Besides, it is hardly an issue if the supply voltage varies (within certain limits): It doesn't make much difference whether the LED current flow is 5 mA or 20 mA (although, for reasons of economics, a low current flow is always better). So, a supply voltage that varies by a factor of four is still more-or-less acceptable. However, if the supply voltage has larger variations, then it is recommended to use a constant-current source.

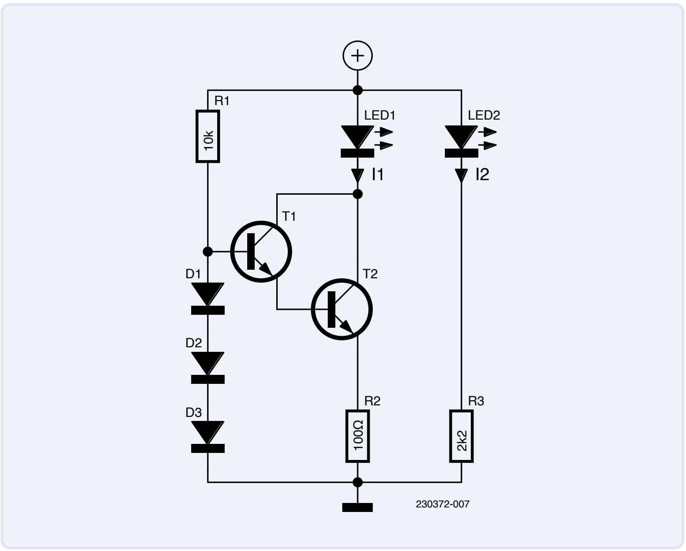


Figure 4: A (more-or-less) constant-current source.

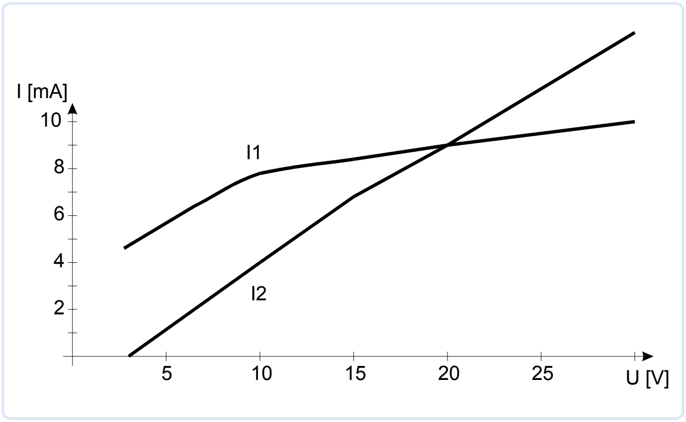


Figure 5: Current flow with and without constant-current source.

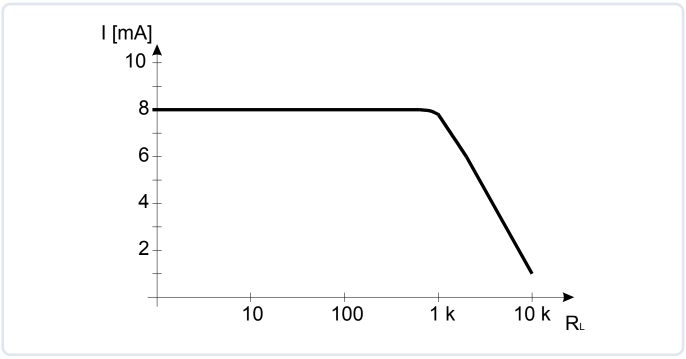


Figure 6: Constant-current source with varying load resistance.

The designation "constant-current source" for the circuit in **Figure 4** is actually a bit of an exaggeration, as can be seen from the graph in **Figure 5**. On the other hand, this circuit makes it possible to use an LED under conditions where the supply voltage can vary by a factor of 40, and that's a good thing, anyway.

Three diodes connected in series provide a more-or-less constant voltage of about 2 V. This voltage is used to drive a Darlington transistor that has an emitter resistance of 100 Ω. The voltage across this resistor is about 0.7 V, resulting in a current flow of about 0.7 mA (which is quite correct in practice). On a side note, a Darlington transistor was not really needed in this example. We measured the current through the LED and plotted it in Figure 5. For comparison, we also measured the current through an LED with only a series resistor and plotted it on the same graph.

In the case of the constant-current source, the current increases noticeably less steeply (the graph is flatter), meaning that the circuit would still be usable at significantly higher supply voltages. The fact that the current is not perfectly constant, which would make the graph horizontal, is due to the voltage increase across the diodes — a Zener diode would provide a significantly better result, and a "two-stage" stabilization would have been even better.

By the way, a constant-current source performs way better when the supply voltage is constant (12 V in **Figure 6**) and the load resistor (where the LED is located) is varied.

At higher load resistance values, the current decreases as otherwise the product of the current and the resistance would be higher than the supply voltage — which is impossible. However, as long as the voltage across the load resistor remains well below the supply voltage, the current through the load is pretty constant.

Of course, you might wonder why we would need a constant current with varying load resistance. Well, without such a current source, the differential amplifier would be impossible — and the differential amplifier (perhaps the most important circuit in electronics) will be discussed in the next installment of this series.

230372-01

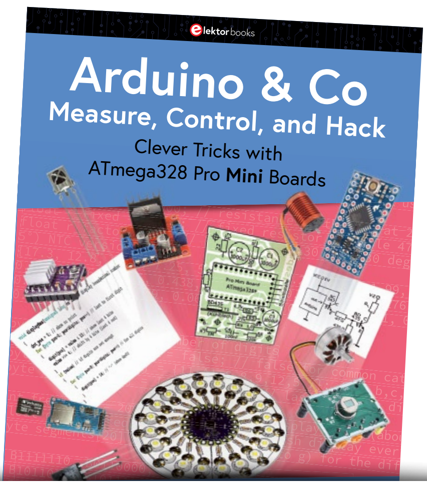
Editor's Note: This series of articles, "Starting Out in Electronics," is based on the book, Basiskurs Elektronik, *by Michael Ebner, which was published in German and Dutch by Elektor.*

Questions or Comments?

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

Infrasound Recorder with the Arduino Pro Mini

A Sample Project from Elektor's "Arduino & Co." Book


elektor books
Arduino & Co
Measure, Control, and Hack
Clever Tricks with ATmega328 Pro Mini Boards

By Robert Sontheimer (Germany)

Let's do something crazy: Let's combine an Arduino Pro Mini (or compatible) with a BMP180 air pressure and an SD card module, and explore long-term infrasound level recording.

Editor's Note: *This article is an excerpt from the 332-page Elektor book,* Arduino & Co — Measure, Control, and Hack*. The excerpt was formatted and lightly edited to match Elektor Magazine's conventions and page layout. Being an extract from a larger publication, this article may refer to discussions elsewhere in the book. The author and editor have done their best to preclude such instances and are happy to help with queries. Contact details are in the* **Questions or Comments?** *box.*

The BMP180 air pressure sensor delivers a maximum of about 100 temperature and air pressure values per second. So, we could generate a stereo audio file (in PCM format with the *.wav* file extension) from this data, over minutes, hours, or days, recording the air pressure on the left channel and the temperature on the right channel. We set the regular playback frequency (sampling rate) to 44,100 Hz, for example. This is a common value that's also used for audio CDs and many other things.

This way, the file will be played back later (e.g. on a laptop or with an audio player) at 441 times the normal speed, and we can hear infrasound! With the original sampling rate of 100 Hz when recording, we can record all frequencies below 50 Hz. An infrasonic sound at 10 Hz becomes 4.41 kHz when played back. 1 Hz becomes 441 Hz, etc. Theoretically, there are no limits below that.

OK, admittedly, the BMP180 is not really suitable as a highly sensitive microphone. Although it already detects pressure deviations at low altitude changes, infrasound has to be quite loud to hear it against the noise during playback.

However, it is much more interesting to use an audio editing program to look at the air pressure and temperature waveforms later — especially if the recording ran over several days. The left channel then shows us the air pressure graph, and the right the temperature graph.

Here, I have simply recorded over almost three days, opened the finished *.wav* file with an audio editing program, and then normalized both channels separately, amplified as much as so that the full range is used. Pictured in **Figure 1**, the upper graph with the pressure readings still looks a bit thick and frayed due to the fact that each pixel on the X axis consists of tens of thousands of measurements, and these also contain some noise and thus fluctuate to a certain degree.

Figure 2 shows the results after processing. The recording was resampled and 4,000 values were averaged into a single value. The noise is gone, and we can see the pressure and temperature curves more clearly.

From the graphs, we can now see a few things: From the pressure graph at top, in terms of time, it became somewhat windier in the second half of the measurement, as many small fluctuations increase there. In a real storm, the fluctuations would be even more extreme.

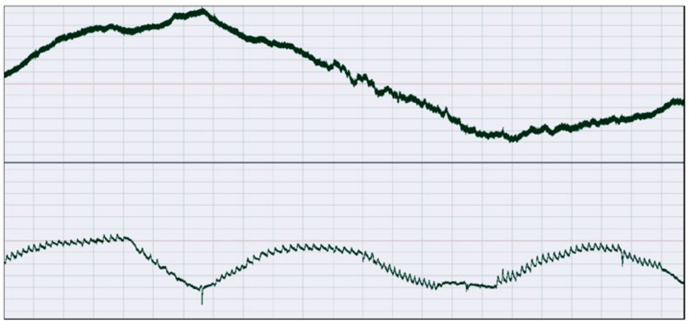


Figure 1: Graph progression for air pressure (above) and temperature (below), essentially unprocessed.

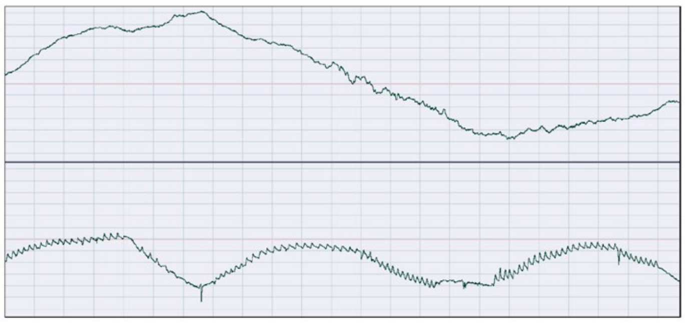


Figure 2: Graph progression of air pressure (above) and temperature (below), processed.

The temperature curve at the bottom clearly shows the day-night rhythm. The almost continuous small spikes are the thermostat switching cycles of the room's heating. One could count how often the thermostat had switched on and off during these three days. Further, one recognizes two quick downward deflections: The window was opened for a short time.

Construction

To implement the project, we need:

- 1 Arduino board (16 MHz ATmega328)
- 1 BMP180 sensor module
- 1 (Micro)SD card module
- 1 push button
- wires or jumpers
- 1 (Micro)SD card, max. 32 GB

Construction is quite simple. In addition to the BMP180 sensor, we need to connect the SD card module, as well as the push button for starting and stopping the recording. Simply follow the connection diagram in **Figure 3**. Soldered wires are always best, but we can also use jumpers. Then, we need two VCC lines, but the Pro Mini has only one VCC pin. There's a very simple solution: Since the BMP180 consumes hardly any power, we can also power it using one output (e.g., with Pin 9 instead of VCC). Then, we only have to define the pin as an output in `setup()` and switch it to High.

Sketch for the Infrasound Recorder

A good section of the sketch named *13.9.ino* written for this project will be presented in partial listings numbered **Listing 1a** through **1h** along with discussions below. The complete program is much longer than can be shown here with the partial listings. The program and subprograms mentioned below are contained in the software companion file released for the book, available for free download from the Elektor Books support website [1]. On the webpage, go to the *Downloads* tab. To be able to follow this extract, please have *13.9.ino* ready for reference.

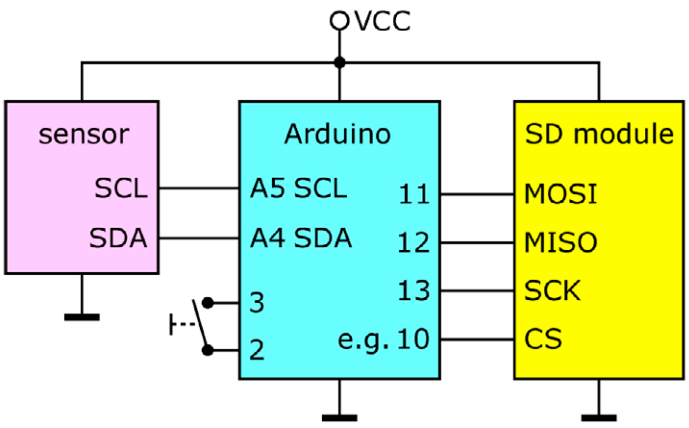


Figure 3: Connection diagram for BMP180 sensor, SD card module, switch, and Arduino.

Listing 1a: Since we combine the air pressure sensor with the SD card module, we have to include three libraries: the I²C interface for the air pressure sensor, the SPI interface for the SD card module, and the special SD card functions.

Then, we define two pins where we can attach a pushbutton to start and stop the recording. Pin 3 is switched to Low and serves as a ground pin for the button. This is very practical, because the button's two pins are then right next to each other. Of course, we could also use the real ground (GND) instead of pin 3.

The other definitions and variables, as well as all program parts from the air pressure sensor and the SD card sketches discussed in the book, are not shown here. We are now only interested in what concerns the recorder, specifically.

Listing 1a

```
#include <Wire.h>     // library for I2C interface
#include <SPI.h>      // library for SPI interface
#include <SD.h>       // library for SD card
#define record_pin 2 // switch pin to ground for
                     //   recording
#define low_pin 3    // Set output pin to LOW
                     //   (used as GND for button)
…
```

Listing 1b: Here, we have the recorder settings. The variable `oversampling` also determines the number of pressure measurements. To record 100 values per second, this setting must be 0. `audio_rate`, on the other hand, specifies how many values per second will be delivered when we later play back the recording as an audio file. `max_file_size` determines the maximum recording length. However, the 10 minutes refers to the playback time — the corresponding recording time is several days. However, we can adjust this value almost arbitrarily. When the maximum file size is reached, the file is closed and a new recording is started.

The `duration` variable determines the delay between measurements.

However, the times here are 500 µs longer because we do not work with delays, but rather with a fixed clock here, during which the Wire instructions for temperature and pressure measurement are also executed.

Listing 1b

```
// 0 -> 1x, 1 -> 2x, 2 -> 4x, 3->8x
char oversampling = 0;
// sampling rate of the output audio file
long audio_rate = 44100;
// 10 minutes' playback time (at 44,100 Hz)
unsigned long max_file_size = 105840044;
// time (µs) according to oversampling
int duration[4] = ;
// time for temperature measurement
int t_duration = 5000;
…
```

Listing 1c: After the special variables for temperature and pressure calculation, which we skipped here again, these variables will be needed later for the recorder.

Listing 1c

```
// system time at which the data are available
unsigned int next_time;

long zero_value;  // pressure zero line
long record_value;// pressure value to save
long zero_temp;   // temperature zero line
long temp_value;  // temperature value to save

// indicates whether recording is activated
boolean must_record = false;
// indicates if recording is currently running
boolean is_recording = false;
// indicates whether pushbutton is pressed
boolean pressed = false;
// actual file size
unsigned long file_size;
// actual file number
unsigned int file_number = 0;
// counts how long button is not pressed anymore
byte state_counter = 0;
File wave_file;   // recording file
```

Listing 1d: `setup()` starts with the definition of the pins for the push button. Due to the active internal pull-up resistor, we do not need any further components.

Listing 1d

```
void setup() {
  // switch pin on input with pull-up
  pinMode(record_pin, INPUT_PULLUP);
  // low pin on output ...
  pinMode(low_pin, OUTPUT);
  // ... and LOW [...]
  digitalWrite(low_pin, LOW);
  …
}
```

Listing 1e: In `loop()`, we now have only the alternating calls of temperature and pressure measurement because in order not to lose time, we do not wait there in each case with a delay for the data, but use the waiting time for all further tasks. For the temperature measurement, function `calculate()` is called while we wait for the result. For the pressure measurement, we call the `recording()` function.

Listing 1e

```
void loop() {
  get_t(); // read temperature
  get_p(); // read pressure
}
```

Listing 1f: Now, the `calculate()` function. The actual temperature and actual pressure are first determined from the measured values, as long as the pressure has already been measured. I have abbreviated this now again with "...", because I'm sure no one wants to read these calculations (with the many formula lines from the datasheet of the BMP180) again.

Then, we check whether the zero baselines for pressure and temperature are already defined. If this is not the case, the first measurement is now defined as the base value. If, on the other hand, no measurement has yet been made, we exit the function completely.

Otherwise, the values are now prepared for recording. For this, we subtract the pressure zero value from the pressure. If the value is overridden, it is limited to the permissible range of -32,768 to 32,767. We then do exactly the same with the temperature. Thus, both values are now ready for recording.

Listing 1f

```
// calculate temperature (in tenth degree) and
// pressure (in pascals)
void calculate() {

  // if pressure has already been measured before
  if (p) {
    …
  }

  // if zero value has not been determined yet
  if (!zero_value) {
    // cancel if no measurement has been taken yet
    if (!pressure) return;
```

```
      // actual value as zero line
      zero_value = pressure;
      // actual value as zero line
      zero_temp = b5;
    }

    // pressure value for recording
    record_value = pressure - zero_value;

    // max value if clipping
    if (record_value > 32767)
      record_value = 32767;
    // negative clipping
    else if (record_value < -32768)
      record_value = -32768;
    // temperature value for recording
    temp_value = b5 - zero_temp;
    // max value if clipping
    if (temp_value > 32767) temp_value = 32767;
    // negative clipping
    else if (temp_value < -32768) temp_
    value = -32768;
  }
```

Listing 1g: In the `recording()` function, two conditions are first checked: whether the recording should run (because it is switched on and has also not yet reached the maximum file size), and whether the recording is actually running. These two distinctions now result in four possibilities. In the first case, the recording should be running, but it is not running yet.

Listing 1g

```
  void recording()  { // recording function
    // if recording should run
    if (must_record && file_size < max_file_size) {
      // if recording is not running yet
      if (!is_recording) {
        …
      }
    }
  }
```

Listing 1h: Here, a new recording is started. For this, the file number is increased in the do-while loop until this (together with the further text) results in a file name that does not exist yet. A new file is then opened with the corresponding file name, and a corresponding message is also output serially. In case of an error, another message is output and the process is stopped using an endless loop.

That exhausts the space allocated to program discussion in this magazine article. Fortunately, the remainder of the program to run the infrasound recorder is well documented just as well as Listings 1a trough 1h shown here.

Listing 1h

```
 // search for the first not yet used
 // recording number
 do {
   file_number++; // next number (starting with 1)
 } // repeat with next number while
    //file already exists

 while (SD.exists("FILE_" + String(file_number)
        + ".wav"));
 Serial.println("Recording no."
                + String(file_number)
                + " is started!");
 wave_file = SD.open("FILE_"
                     + String(file_number)
                     + ".wav", FILE_WRITE);

 if (!wave_file) { // if file could not be opened
    Serial.println
       ("Error: File could not be opened!");
 // do not continue (infinite loop)
     while (true);
 }
```

Usage

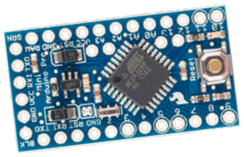
At the beginning of the sketch, we can set three variables:

- `oversampling` should be set to 0 to use the fastest recording frequency of 100 Hz.
- `audio_rate` specifies the later playback frequency. There, we can also use a value lower than 44,100, so that the time factor (44,100 / 100 = 441) is not quite so high. Common values are 8,000, 11,025, 16,000, 22,050, 32,000, 44,100, and 48,000. Not every player can play other frequencies.
- with `max_file_size`, we define the maximum file size. 105,840,044 corresponds to a playback time of 10 minutes at 44,100 Hz, but from a recording time of more than three days.

We should then "normalize" the completed recording, that is, increase the volume of the signal so that the full range is used, but without exceeding the maximum. Without normalization, the signal will be too quiet. Of course, we could add a gain factor when recording, but we don't know beforehand how much the temperature and air pressure will vary during recording. Normalization can be done with audio editing programs in a few clicks. It should be done separately for both channels. With such a program, we can then also view the recording as measurement graphs. The left channel shows the course of the air pressure during recording, while the right channel shows the temperature.

Weather Recorder

If we change the `oversampling` value to 3, only a good 32 (instead of 100) values per second will be recorded, which is still far more than enough than we need for weather recordings. We could also



combine the altimeter (from section 13.8 in our book) with the SD card functions so we can write a corresponding text line into a file for each serial output as well. This way, we get the weather recording as a text file.

So, you see, there are countless possibilities for what you can do, and the individual book chapters offer — besides the concrete projects — a lot of information and suggestions, which should help you in implementing as many of your own ideas as possible.

Unfortunately, we have now reached the end of this article, and I wish all readers a lot of fun in measuring, controlling, and hacking!

230393-01

Questions or Comments?

Do you have any questions or comments related to this article? Email the author at r.sont@freenet.de or Elektor at editor@elektor.com.

About the Author

Robert Sontheimer was immediately on board when the first home computers arrived in our living rooms some 40 years ago, with his ZX81 and C64. Back then, he converted a plotter into a scanner, among other quirky and original ideas, and today he uses Arduino Pro Minis to control entire CNC laser machines and has even invented a matching suction system: his "self-changing toilet paper filter." In his office, he has a magnet that's been levitating for years – controlled by a Pro Mini, of course.

WEB LINK

[1] Software Archive for the Book: https://elektor.com/20243

Any Signal.
Anytime.
Anywhere.

SHA850A

Handheld Spectrum Analyzer

Frequency Range: 9 kHz ~ 7.5 GHz

Visit us at
booth 410C

SIGLENT®

www.siglenteu.com
Info-eu@siglent.com

Cloud-Based Energy Meter

With ESP32 Module and PZEM-004T Voltage/Current Sensor

A project from Elettronica In
https://elettronicain.it

By Emanuele Signoretta (Italy)

With the ever-increasing price of electricity, wise use and savings have become a must. With an ESP32 module and a few other hardware components, it is possible to create an energy meter that sends our energy data to the InfluxDb Cloud platform via a connection on the Wi-Fi network.

Italy, like many other countries, relies heavily on foreign energy sources because its own renewable energy production is not enough to meet national demand. This becomes a bigger problem when external factors limit the availability of fossil fuel sources, leading to higher prices. As a result, people in Italy, just like us, have to cut down on energy consumption, sacrifice certain comforts, and closely monitor their energy usage to avoid unpleasant surprises on their electricity and gas bills. The primary motivation for reducing energy consumption is the fear of overspending, rather than concerns about climate change.

To assist those who are conscious of electricity usage, this article introduces an energy consumption meter that utilizes an Espressif ESP32 module and a PZEM-004T voltage/current sensor. The meter collects data and sends it to the cloud, specifically the servers of the InfluxDB online service.

The Hardware

For the hardware component of the energy meter, we need an ESP32 Wi-Fi module (**Figure 1**) and a PZEM-004T sensor (seen encapsulated in its transparent plastic case in **Figure 2**). We also need a switching power supply with a 5 V output (if you choose one with a Micro USB output, you can use it to power the ESP board and get power for the PZME-004T from V_{IN}), female-female jumpers for the Arduino, terminals, some electrical cable, and a plastic box [2]. The sensor datasheet can be found at [3], while **Figure 3** shows the circuit's overall wiring — we'll go into more detail on that under *Installation* below.

The module comprises a circuit for sensing current and voltage, housed in a plastic case with an open toroid. It incorporates minimal

Figure 1: The ESP32 board.

Figure 2: The PZEM-004T sensor with transformer.

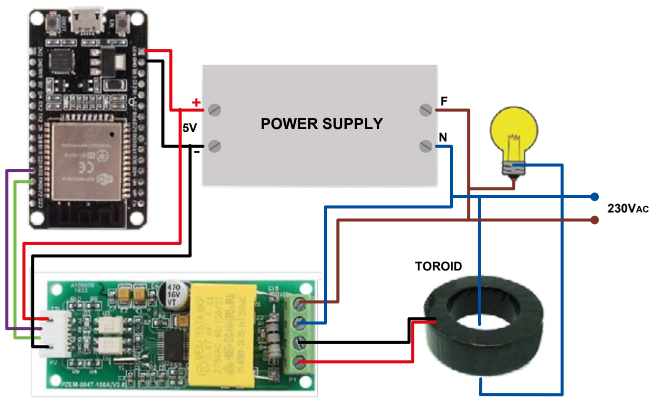
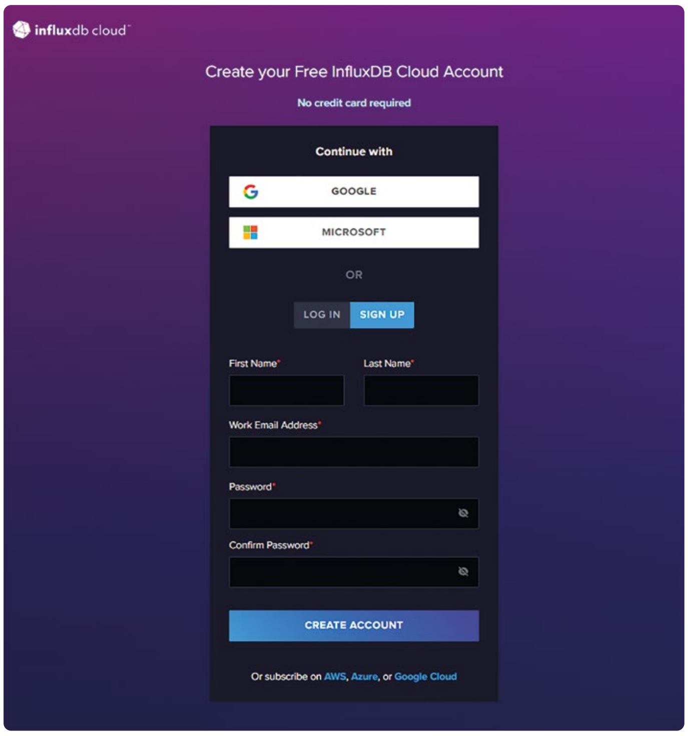


Figure 3: The circuit's wiring diagram.

Figure 4: InfluxDB logo.



Create your Free InfluxDB Cloud Account

No credit card required

Continue with

GOOGLE

MICROSOFT

OR

LOG IN SIGN UP

First Name*

Last Name*

Work Email Address*

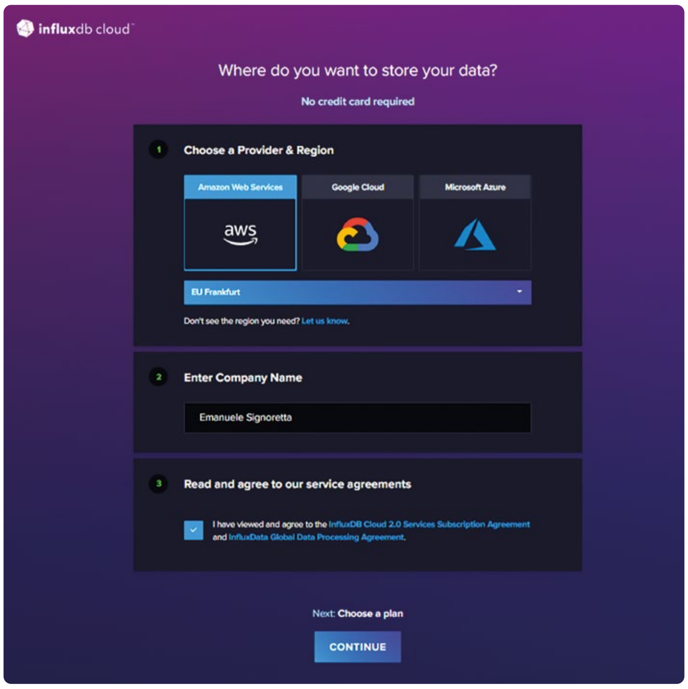
Password*

Confirm Password*

CREATE ACCOUNT

Or subscribe on AWS, Azure, or Google Cloud

Figure 5: InfluxDB registration page.



Where do you want to store your data?

No credit card required

1 Choose a Provider & Region

Amazon Web Services | Google Cloud | Microsoft Azure

EU Frankfurt

Don't see the region you need? Let us know.

2 Enter Company Name

Emanuele Signoretta

3 Read and agree to our service agreements

✓ I have viewed and agree to the InfluxDB Cloud 2.0 Services Subscription Agreement and InfluxData Global Data Processing Agreement.

Next: Choose a plan

CONTINUE

Figure 6: Choice of provider and acceptance of terms of service.

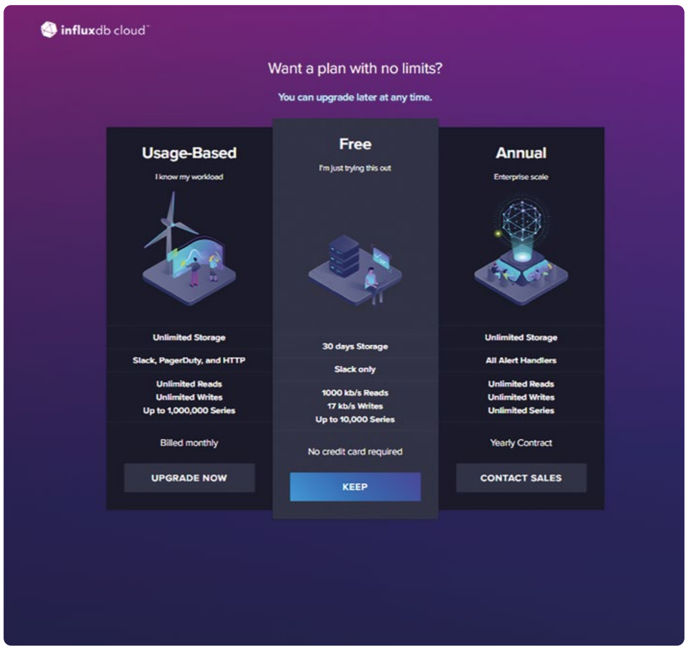
electronics, isolated by optocouplers, and has a serial interface. The module can accurately measure voltages ranging from 80 V to 260 V, with a resolution of 0.1 V and an accuracy of 0.5%. Similarly, it can measure currents from 0 A to 100 A with an accuracy of 0.5% and a resolution of 0.001 A. Additionally, the sensor can determine the phase angle (power factor) between voltage and current vectors with 1% accuracy, providing readings for active and reactive power. This capability allows us to evalute the electrical efficiency of the device being measured. The selection of circuit components was based on cost-effectiveness, considering the ongoing semiconductor crisis, and the ESP32's impressive specifications.

InfluxDB Configuration

Multiple software packages are available for receiving and analyzing data transmitted by IoT devices. Notable options include Home Assistant, Grafana, Blynk, Thingspeak, and more. In this context, we will focus on InfluxDB, whose logo is shown in **Figure 4**.

InfluxDB is versatile software that provides functionality such as creating dashboards, running queries, and sending alerts. It offers multiple deployment options, including executable versions for various architectures, Docker containers, and cloud-based solutions. Although initially we intended to install an instance on a Raspberry Pi for data ownership reasons, the scarcity of components forced us to adopt the InfluxDb Cloud v2 service instead.

To get started, please visit link [4] and proceed with registration. You can choose to sign up using your Microsoft or Google credentials, or manually fill in the required fields as shown in **Figure 5**. After completing the registration process, you will be presented with a screen resembling **Figure 6**, where you will be prompted to provide various details,

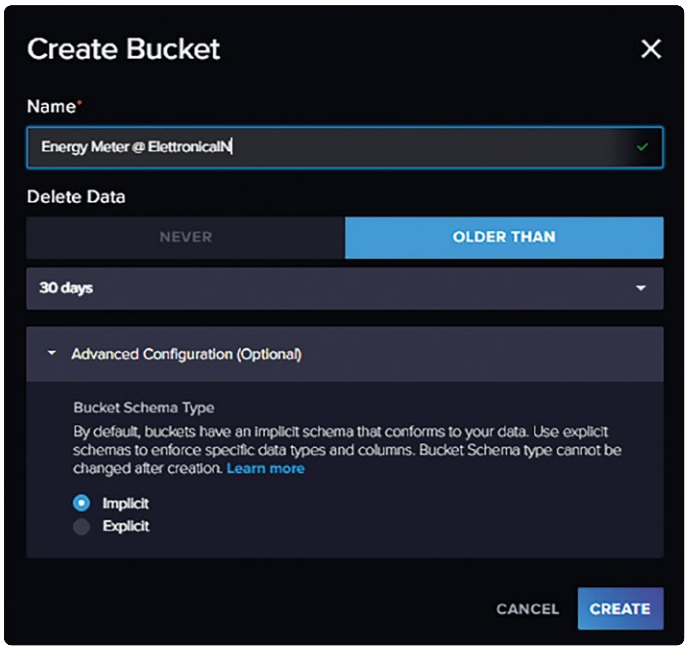

influxdb cloud

Want a plan with no limits?

You can upgrade later at any time.

Usage-Based	Free	Annual
I know my workload	I'm just trying this out	Enterprise scale
Unlimited Storage	30 days Storage	Unlimited Storage
Slack, PagerDuty, and HTTP	Slack only	All Alert Handlers
Unlimited Reads Unlimited Writes Up to 1,000,000 Series	1000 kb/s Reads 17 kb/s Writes Up to 10,000 Series	Unlimited Reads Unlimited Writes Unlimited Series
Billed monthly	No credit card required	Yearly Contract
UPGRADE NOW	KEEP	CONTACT SALES

Figure 7: Choice of subscription plan.


Create Bucket

Name*: Energy Meter @ ElettronicaIN

Delete Data: NEVER [✓ OLDER THAN]

30 days

Advanced Configuration (Optional)

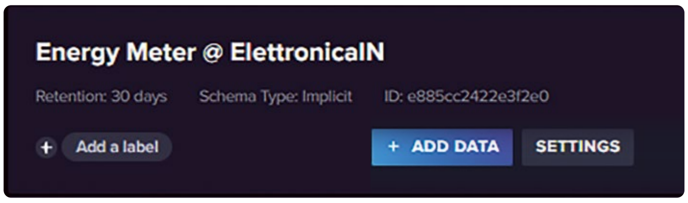
Bucket Schema Type

By default, buckets have an implicit schema that conforms to your data. Use explicit schemas to enforce specific data types and columns. Bucket Schema type cannot be changed after creation. Learn more

[✓ Implicit] Explicit

CANCEL CREATE

Figure 8: Creating a new bucket.


Energy Meter @ ElettronicaIN

Retention: 30 days Schema Type: Implicit ID: e885cc2422e3f2e0

Add a label + ADD DATA SETTINGS

Figure 9: A newly created bucket.

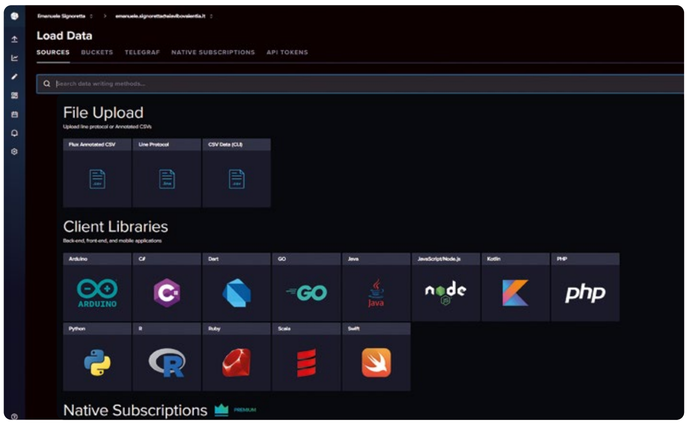
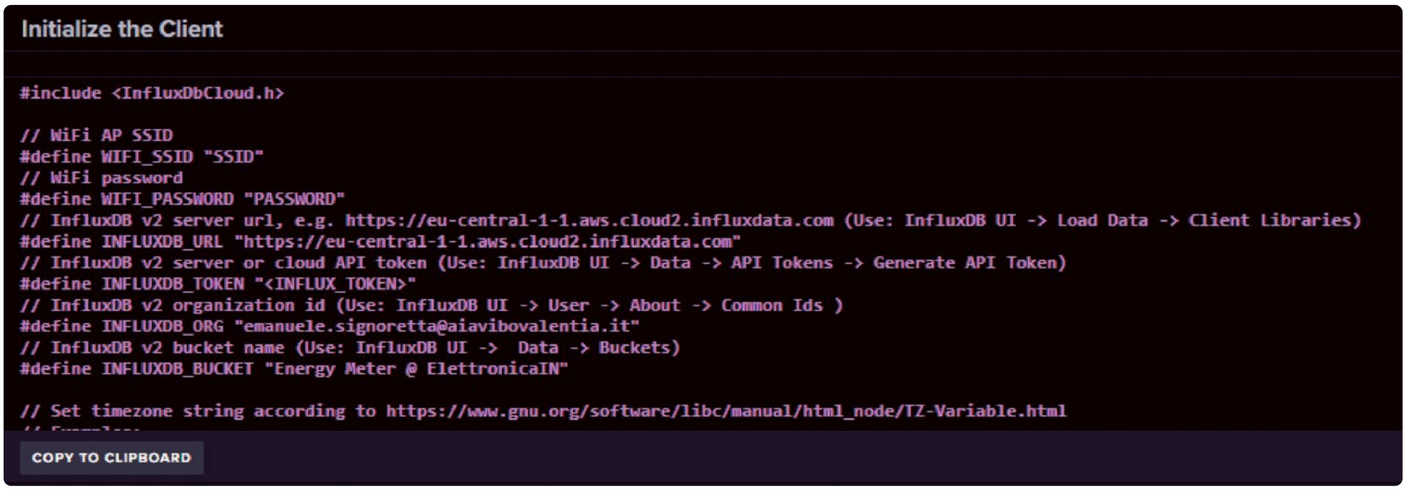


Figure 10: Selection of data upload source.


Initialize the Client

```
#include <InfluxDbCloud.h>

// WiFi AP SSID
#define WIFI_SSID "SSID"
// WiFi password
#define WIFI_PASSWORD "PASSWORD"
// InfluxDB v2 server url, e.g. https://eu-central-1-1.aws.cloud2.influxdata.com (Use: InfluxDB UI -> Load Data -> Client Libraries)
#define INFLUXDB_URL "https://eu-central-1-1.aws.cloud2.influxdata.com"
// InfluxDB v2 server or cloud API token (Use: InfluxDB UI -> Data -> API Tokens -> Generate API Token)
#define INFLUXDB_TOKEN "<INFLUX_TOKEN>"
// InfluxDB v2 organization id (Use: InfluxDB UI -> User -> About -> Common Ids )
#define INFLUXDB_ORG "emanuele.signoretta@aiavibovalentia.it"
// InfluxDB v2 bucket name (Use: InfluxDB UI ->  Data -> Buckets)
#define INFLUXDB_BUCKET "Energy Meter @ ElettronicaIN"

// Set timezone string according to https://www.gnu.org/software/libc/manual/html_node/TZ-Variable.html
```

COPY TO CLIPBOARD

Figure 11: Snippet of a demo sketch.

including the service provider name. For our project, we have selected Amazon Web Services (AWS) as the provider.

On the subsequent screen (depicted in **Figure 7**), you will need to select a usage plan. In our case, we opted for the free plan, which grants data storage for a duration of 30 days and the ability to receive notifications via Slack.

After confirming our selection, the personal dashboard will be displayed. To create a bucket, navigate to *Load Data → Buckets → Create new bucket*. This action will open a screen similar to the one depicted in **Figure 8**. Fill in the *Name* field and click on the *Create* button. Once the bucket is successfully created, it will appear among the available data sources, as shown in **Figure 9**.

Next, click on *Add data → Client library*. A new screen will appear, offering various options for uploading data. From these alternatives, select *Arduino* (**Figure 10**). In the subsequent window (**Figure 11**), you will see a series of code snippets that constitute a demonstration sketch. Copy the data provided for `INFLUXDB_URL`, `INFLUXDB_ORG`, and `INFLUXDB_BUCKET` from the first snippet.

To complete the setup, we need to generate an access token for the bucket. To achieve this, access the left-hand sidebar and navigate to *Load Data → API Tokens*.

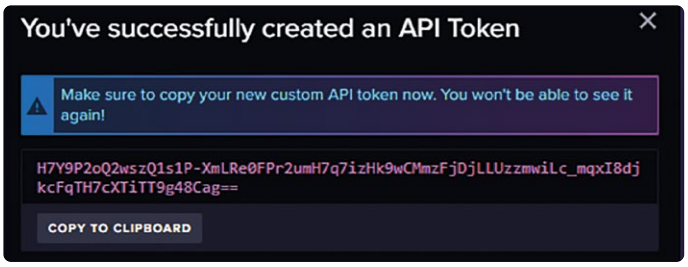
In the newly opened window, click on *Generate API Token*. After selecting the bucket, enable both read and write permissions, as depicted in **Figure 12**. Once the token is generated, copy it and keep it safe, as we will need it in the upcoming steps.

The Sketch

The sketch for our project is a combination of several sketches from the libraries we used. You can download the sketch files from the GitHub repository [5]. Now, let's analyze our code, which is divided into three sections that we will present with corresponding listings.

First, let's discuss the inclusion of necessary libraries and dependencies, as shown in **Listing 1**.

Regarding this, it's important to note that the *WiFiMulti* library is utilized for managing Wi-Fi communication between the ESP32 and the access point. The *ESPmDNS*, *WiFiUDP*, and *ArduinoOTA* [6]



You've successfully created an API Token

Make sure to copy your new custom API token now. You won't be able to see it again!

```
H7Y9P2oQ2wszQ1s1P-XmLRe0FPr2umH7q7izHk9wCMmzFjDjLLUzzmwiLc_mqxI8dj
kcFqTH7cXTiTT9g48Cag==
```

Figure 12: Creating an API key.

Listing 1: Including the libraries.

```
#include <WiFiMulti.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
#include <InfluxDbClient.h>
#include <InfluxDbCloud.h>
#include <PZEM004Tv30.h>
#include <Every.h>
```

libraries are employed for OTA (over-the-air) uploading of sketches and host-name management. Please make sure to have Python version 2.7.x installed on your PC for this functionality. The *InfluxDbClient* and *InfluxDbCloud* libraries are used for data upload to the InfluxDB Cloud. The *PZEM004Tv30* library [7] facilitates serial communication with the sensor. Lastly, the *Every* library allows the execution of code blocks at regular intervals without relying on `delay()`.

Now, let's examine the subsequent code portions, starting with **Listing 2**, which contains the pre-processor section. By using `#define REFRESH_TIME 5000`, we set the interval (in milliseconds) for uploading data to the cloud. We then define the device name, serial communication pins, and the desired serial port. Objects related to the Wi-Fi network and the sensor are instantiated. IP addresses are defined, and if you wish to configure a connection to the access point with a static IP, you can adjust the parameters according to your network's subnet. Lastly, we define parameters for Wi-Fi connection and InfluxDB Cloud access. Replace these lines of code with the copied snippet and enter the missing data.

By using `#define WIFI_SSID` and `#define WIFI_PASSWORD`, we specify the Wi-Fi network's SSID and passphrase, respectively.

Next, we set the parameters for connecting to InfluxDB Cloud. The only missing parameter is `INFLUXDB_TOKEN`, which can be retrieved from the API key we created earlier. Insert the API key between the quotation marks. With `#define TZ_INFO "CET-1CEST,M3.5.0,M10.5.0/3"`, we specify the Central European Time zone for timestamps.

To create the client object for connecting to the InfluxDB server, use the following code:

```
InfluxDBClient client(INFLUXDB_URL, INFLUXDB_ORG,
                      INFLUXDB_BUCKET, INFLUXDB_TOKEN,
InfluxDbCloud2CACert);
```

Finally, the sensor object is created with `Point sensor("EnergyMeter")`. This object, named `EnergyMeter`, will be associated with all the data collected by the sensor. The code in **Listing 3** represents the board setup and initializes the serial port at a baud rate of 115,200.

Listing 2: Definitions

```
#define REFRESH_TIME 5000 // Delay interval for data upload
#define DEVICE "ESP32"
#define PZEM_RX_PIN 27
#define PZEM_TX_PIN 26
#define PZEM_SERIAL Serial2

/*************************
ESP32 initialization
---------------------

   The ESP32 HW Serial interface can be routed to any GPIO pin
   Here we initialize the PZEM on Serial2 with RX/TX pins 26 and 27
*/
PZEM004Tv30 pzem(PZEM_SERIAL, PZEM_RX_PIN, PZEM_TX_PIN);
WiFiMulti wifiMulti;
IPAddress local_IP(192, 168, 178, 154);
IPAddress gateway(192, 168, 178, 1);
IPAddress subnet(255, 255, 255, 0);
IPAddress primaryDNS(192, 168, 178, 1); //optional
IPAddress secondaryDNS(1, 1, 1, 1); //optional
// WiFi AP SSID
#define WIFI_SSID ""
// WiFi password
#define WIFI_PASSWORD ""
// InfluxDB v2 server url, e.g. https://eu-central-1-1.aws.cloud2.influxdata.com
 // (Use: InfluxDB UI -> Load Data -> Client Libraries)
#define INFLUXDB_URL "https://eu-central-1-1.aws.cloud2.influxdata.com"
// InfluxDB v2 server or cloud API token
 //   (Use: InfluxDB UI -> Data -> API Tokens -> Generate API Token)
#define INFLUXDB_TOKEN ""
// InfluxDB v2 organization id (Use: InfluxDB UI -> User -> About -> Common Ids )
#define INFLUXDB_ORG ""
// InfluxDB v2 bucket name (Use: InfluxDB UI ->  Data -> Buckets)
#define INFLUXDB_BUCKET "Energy Meter @ ElettronicaIN"
// Set timezone string according to
//    https://www.gnu.org/software/libc/manual/html_node/TZ-Variable.html
#define TZ_INFO "CET-1CEST,M3.5.0,M10.5.0/3"
// InfluxDB client instance with preconfigured InfluxCloud certificate
InfluxDBClient client(INFLUXDB_URL, INFLUXDB_ORG,
    INFLUXDB_BUCKET, INFLUXDB_TOKEN, InfluxDbCloud2CACert);
// Data point
Point sensor("EnergyMeter");
```

Listing 3: setup()

```
void setup() {
  Serial.begin(115200);
  // Uncomment in order to reset the internal energy counter
  // pzem.resetEnergy()
  // Setup wifi
  WiFi.mode(WIFI_STA);
  //Comment to use DHCP instead of static IP
   if (!WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS)) {
    Serial.println("STA Failed to configure");
  }
  wifiMulti.addAP(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to wifi");
  while (wifiMulti.run() != WL_CONNECTED) {
    //Serial.print(".");
    //delay(100);
    Serial.println("Connection Failed! Rebooting...");
    delay(5000);
    ESP.restart();
  }
  Serial.println();
  // Port defaults to 3232
  // ArduinoOTA.setPort(3232);
  // Hostname defaults to esp3232-[MAC]
  ArduinoOTA.setHostname("ESP32-Energy Meter");
  // No authentication by default
  // ArduinoOTA.setPassword("admin");
  // Password can be set with it's md5 value as well
  // MD5(admin) = 21232f297a57a5a743894a0e4a801fc3
  // ArduinoOTA.setPasswordHash("21232f297a57a5a743894a0e4a801fc3");
  ArduinoOTA
  .onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH)
      type = "sketch";
    else // U_SPIFFS
      type = "filesystem";
    // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end()
    Serial.println("Start updating " + type);
  })
  .onEnd([]() {
    Serial.println("\nEnd");
  })
  .onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
  })
  .onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
    else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
    else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
    else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
    else if (error == OTA_END_ERROR) Serial.println("End Failed");
  });
  ArduinoOTA.begin();
  // Add tags
  sensor.addTag("Dispositivo", DEVICE);
  // Accurate time is necessary for certificate validation and writing in batches
  // For the fastest time sync find NTP servers in your area: https://www.pool.ntp.org/zone/
  // Syncing progress and the time will be printed to Serial.
  timeSync(TZ_INFO, "pool.ntp.org", "time.nis.gov");\
  // Check server connection
  if (client.validateConnection()) {
    Serial.print("Connected to InfluxDB: ");
    Serial.println(client.getServerUrl());
  } else {
    Serial.print("InfluxDB connection failed: ");
    Serial.println(client.getLastErrorMessage());
  }
}
```

The code block begins with WiFi.mode(WIFI_STA); which initializes Wi-Fi in station mode for connecting to an access point. The next block of code:

```
if (!WiFi.config(local_IP, gateway, subnet,
  primaryDNS, secondaryDNS)) {
 Serial.println("STA failed to configure");
}
```

sets a static IP instead of using DHCP. If there is an error, a warning message is printed to the serial port. If you want to use DHCP, you can comment out this part of the code.

The `wifiMulti.addAP(WIFI_SSID, WIFI_PASSWORD)` function call attempts to connect to the access point using the specified SSID and passphrase.

A connection to the AP is then attempted, and, if it fails, an error is first printed to the serial port, and the board is restarted after a wait of five seconds.

The hostname is set with `ArduinoOTA.setHostname("ESP32-Energy Meter")`, which will be transmitted via mDNS and displayed on the Arduino IDE for loading the OTA sketch. There is commented code for restricting OTA sketch loading to users with a username and password. Uncomment these lines of code to add this security feature.

The OTA mode settings are managed with the following code block:

```
ArduinoOTA.onStart([]() {
  String type;
  // …
  if (error == OTA_AUTH_ERROR) Serial.println("Auth
Failed");
  else if (error == OTA_BEGIN_ERROR) Serial.println("Begin
Failed");
   else if (error == OTA_CONNECT_ERROR) Serial.
println("Connect Failed");
   else if (error == OTA_RECEIVE_ERROR) Serial.
println("Receive Failed");
  else if (error == OTA_END_ERROR) Serial.println("End
Failed");
});
ArduinoOTA.begin();
```

The code block handles OTA start, end, sketch loading, and errors in that order. `ArduinoOTA.begin()` starts the OTA loading process.

With `sensor.addTag("Device", DEVICE);` a *tag* is set, which in this case corresponds to the device name. This feature could be useful to distinguish one or more sensors within a fleet.

With `timeSync(TZ_INFO, "pool.ntp.org", "time.nis.gov");` the ESP connects to the two NTP (network time protocol) servers *pool.ntp.org* and *time.nis.gov* to obtain the date and time.

Finally, the code block attempts to connect to the InfluxDB servers:

```
if (client.validateConnection()) {
  Serial.print("Connected to InfluxDB: ");
  Serial.println(client.getServerUrl());
}
else {
  Serial.print("InfluxDB connection failed: ");
  Serial.println(client.getLastErrorMessage());
}
```

If the connection is successful, the server URL is printed to the serial port. If not, the corresponding error message is printed, instead.

Now, let's move on to **Listing 4**, which contains the `loop()` code for our sketch. Using `ArduinoOTA.handle()`, the ESP32 waits for OTA to receive any compiled sketches.

With `EVERY(REFRESH_TIME) { … }` the code within the brackets is executed at regular intervals and without using delays. The time interval has been previously defined. Within the `EVERY` block, the address of the PZEM is first read and printed out in serial.

The code within the `EVERY(REFRESH_TIME) { … }` block is executed at the regular intervals defined earlier, without using delays. Within this block, the PZEM's address is read and printed to the serial port.

Next, sensor variables are created and initialized using functions from the sensor library. These variables include voltage, current, active power (in Wh), active energy (in kWh), frequency, and power factor (a ratio between -1 and 1, representing the ratio between active power and apparent power). If any of the variables are not a number, a read error is printed to the serial port. Otherwise, the sensor readings are printed. After this, the initialized fields and timestamps are cleared, and the variables are prepared for uploading to the cloud. Finally, the Wi-Fi connection and successful data upload are checked. If the upload fails, the last error code is printed to the serial port.

Installation

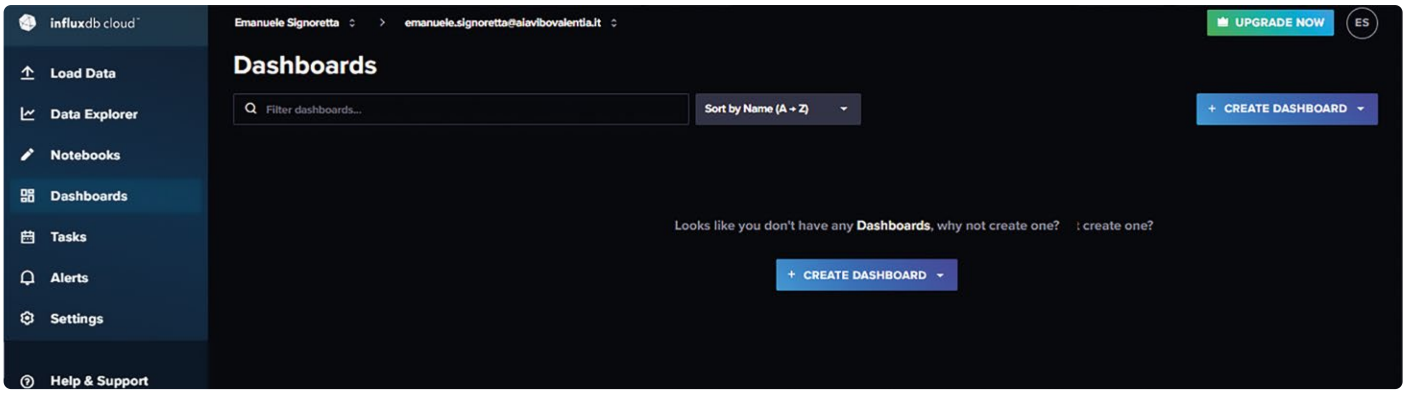
We planned to enclose the power supply, ESP32 and the main sensor module inside a plastic box, with only the wires for the power supply and those going to the current sensor toroid going into the board. If you want to measure the overall energy consumption of your house, the power input of our energy metering system must be connected as close as possible to the AC line coming out of the house's electrical panel. Otherwise, as illustrated in Figure 3, it is always possible to measure the energy absorbed by a single electrical appliance of any type. The hinged opening of the toroid makes it very convenient to clamp the electrical cable under measurement down. In the case of three-core power cables (Live, Neutral and Earth), the outer insulation sleeve must be carefully cut lengthwise (**be sure to unplug it first, please!**),

Listing 4: loop()

```
void loop() {

  ArduinoOTA.handle();

  EVERY(REFRESH_TIME) {
    // Print the custom address of the PZEM
    Serial.print("Custom Address:");
    Serial.println(pzem.readAddress(), HEX);
    // Read the data from the sensor
    float voltage = pzem.voltage();
    float current = pzem.current();
    float power = pzem.power();
    float energy = pzem.energy();
    float frequency = pzem.frequency();
    float pf = pzem.pf();
    // Check if the data is valid
    if (isnan(voltage)) {
      Serial.println("Error reading voltage");
    } else if (isnan(current)) {
      Serial.println("Error reading current");
    } else if (isnan(power)) {
      Serial.println("Error reading power");
    } else if (isnan(energy)) {
      Serial.println("Error reading energy");
    } else if (isnan(frequency)) {
      Serial.println("Error reading frequency");
    } else if (isnan(pf)) {
      Serial.println("Error reading power factor");
    } else {
      // Print the values to the Serial console
      Serial.print("Voltage: ");      Serial.print(voltage);      Serial.println("V");
      Serial.print("Current: ");      Serial.print(current);      Serial.println("A");
      Serial.print("Power: ");        Serial.print(power);        Serial.println("W");
      Serial.print("Energy: ");       Serial.print(energy, 3);     Serial.println("kWh");
      Serial.print("Frequency: ");    Serial.print(frequency, 1); Serial.println("Hz");
      Serial.print("PF: ");           Serial.println(pf);
      ////UPLOAD DATA
      // Clear fields for reusing the point. Tags will remain untouched
      sensor.clearFields();
      // Store measured value into point
      sensor.addField("Tensione", voltage);
      sensor.addField("Corrente", current);
      sensor.addField("Potenza", power);
      sensor.addField("Energia", energy);
      sensor.addField("Frequenza", frequency);
      sensor.addField("Fattore di potenza", pf);
      // Print what are we exactly writing
      Serial.print("Writing: ");
      Serial.println(sensor.toLineProtocol());
      // Check WiFi connection and reconnect if needed
      if (wifiMulti.run() != WL_CONNECTED) {
        Serial.println("Wifi connection lost");
      }
      // Write point
      if (!client.writePoint(sensor)) {
        Serial.print("InfluxDB write failed: ");
        Serial.println(client.getLastErrorMessage());
      }
    }
    Serial.println();
  }
}
```



Looks like you don't have any **Dashboards**, why not create one?

Figure 13: Creating the dashboard.

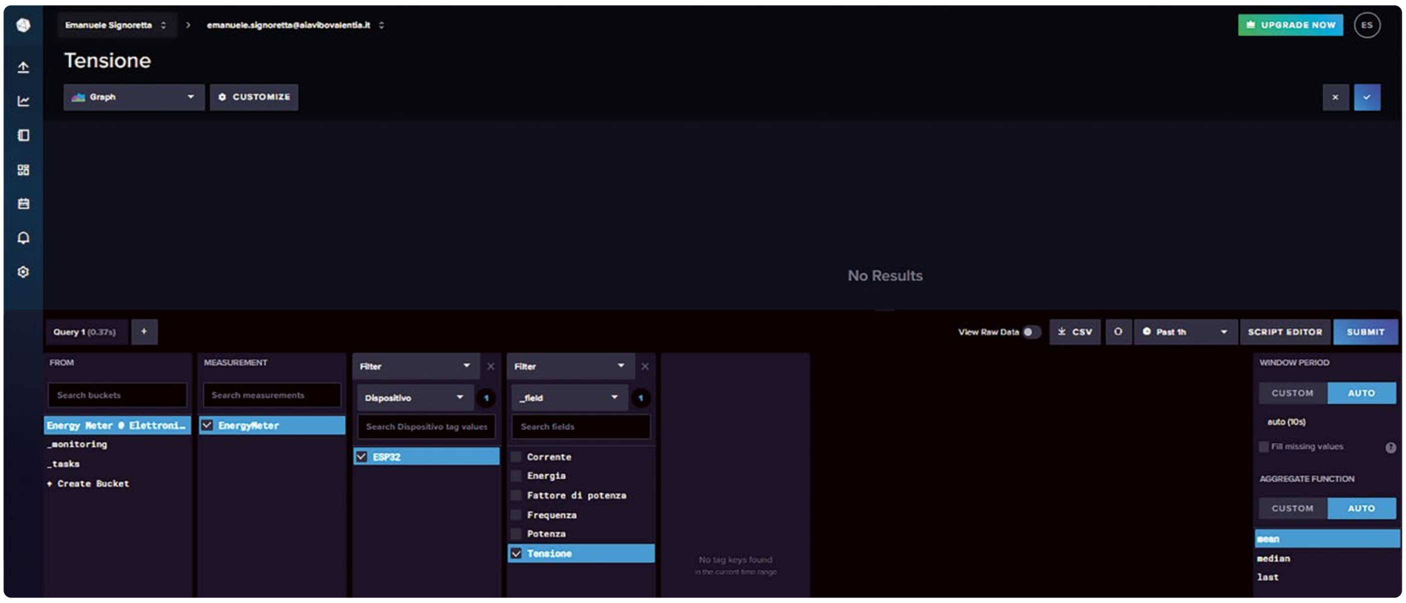


Figure 14: Adding the dashboard graphics.

without affecting the insulation of the three inner cables. They will be separated and only the Live wire (normally black or brown) will be placed inside the measuring toroid.

To install the energy meter, simply follow the diagram already shown at the beginning of this article, in Figure 3.

In particular, you must connect two wires in parallel to the input of the power supply and between pins N and L of the PZEM-004T. **Remember to always be very careful because you are working on the 230 V(AC) mains wires!**

Next, we release the toroid safety catch and run one of the two wires going to the electrical system, then connect the toroid wires to the CT pins of the sensor. Additionally, we connect the sensor's main module to the ESP32, according to **Table 1**. To power the ESP, you can use a power supply with a Micro USB output, taking the 5 V for the sensor from VIN and GND. Once the sketch is loaded, we can close all the electronics in the box and start the system.

Startup and Dashboard Setup

Once the board has started up, we go back to the InfluxDB website and, from our dashboard, we go to *Dashboards → Create new dashboard*. Doing so will bring up a screen like the one shown in **Figure 13**, where we assign a name to our dashboard and click on *Add Cell*. In the new screen that will appear (**Figure 14**), we are going to select which parameters to include in each of the charts that make up the dashboard. We select, in order, the bucket, the measurements, the device, and finally the values to be displayed. You can choose which graphics to use for the values: heatmap, gauge, simple graph, table, etc., and also which scaling values to utilize. We customize the cells to our liking and repeat the procedure for each value we wish to display until we obtain a result like the one visible in **Figure 15**, which illustrates our complete dashboard.

Table 1: Connections between the Energy Meter and the ESP32 modules.

PZEM-004 pin	ESP32 pin
VCC	V5
GND	GND
RX	26
TX	27

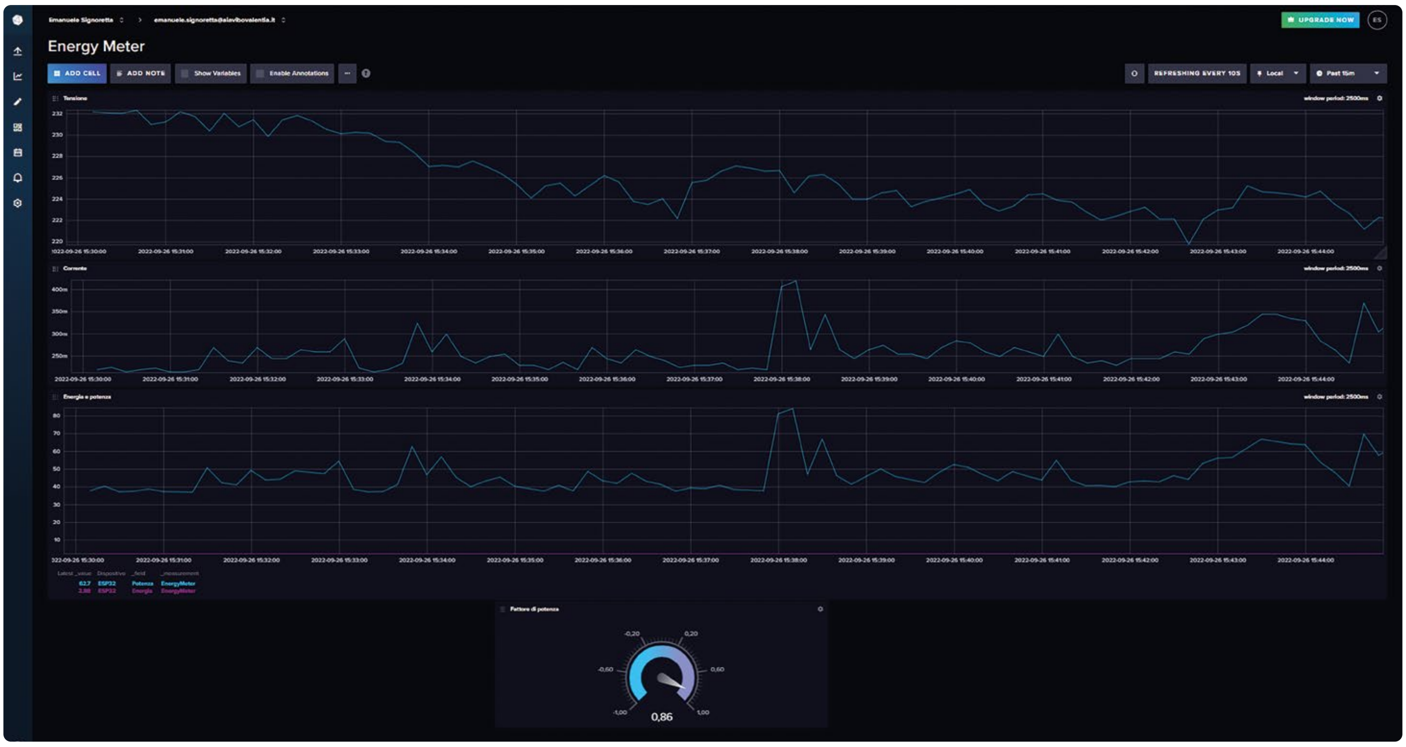


Figure 15: The completed dashboard.

Conclusion

This concludes the description of our simple but powerful energy meter. The flexibility of the ESP32 board and the many features of the supported network protocols and platforms allow us to expand its fields of application according to our needs, perhaps by integrating the meter into a load-management system, and still be capable of autonomous actions.

230279-01

Questions or Comments?

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

About the Author

Emanuele Signoretta was born in 2000 in Vibo Valentia, a small city in the south of Italy. He is passionate about all the stuff concerning ICT. While attending High School, he discovered Arduino and its programming simplicity. Starting there, he wrote code for Arduino and Fishino boards. Now, he's started with STM32- and ESP32-based boards. He believes in the open-source ethos and uses Linux-based distros. Emanuele is currently completeing his Bachelor's Degree in Electronic and Communication Engineering at Politecnico di Torino. Furthermore, he works for Rai, the Italian national broadcaster.

WEB LINKS

[1] ESP32 board: https://futuranet.it/prodotto/esp32-scheda-di-sviluppo-32-gpio-con-wifi-e-bluetooth
[2] Plastic box: https://futuranet.it/prodotto/contenitore-plastico-ermetico-546x784x1182-mm
[3] PZEM-004 Datasheet on Github: https://bit.ly/3qTZdHf
[4] InfluxDB Account: https://cloud2.influxdata.com/signup
[5] The project's GitHub repository: http://github.com/signorettae/ESP32-EnergyMeter
[6] ArduinoOTA Library: https://github.com/jandrassy/ArduinoOTA
[7] Arduino library for the Updated PZEM-004T: http://github.com/mandulaj/PZEM-004T-v30

A Bare-Metal Programming Guide (Part 2)

Accurate Timing, the UART, and Debugging

By Sergey Lyubka (Ireland)

In the first part of the guide, we learned how to access microcontroller registers to control pins. Additionally, we created minimal firmware and our first blinking LED demo with the help of a linker script and a Makefile. In this installment of the series, we deal with accurate timing via system clocks, the UART, and debugging.

Editor's note: This guide is a living document on GitHub [1] and it grows. So, we decided to extend this series by an additional installment, which you will find in the next edition of Elektor (11-12/2023).

Blinky with Systick Interrupt

For our first LED "Blinky" demo, we used a delay function called `spin()` that just executed NOP instructions a given number of times (see the first part of the series at [2]).

In order to implement much more accurate timekeeping, we should enable ARM's SysTick interrupt. SysTick is a 24-bit hardware counter, and is part of the ARM core, therefore it is documented by the Arm® v7-M Architecture Reference Manual [3]. Looking at this manual, we see that SysTick has four registers:

- CTRL — used to enable/disable SysTick
- LOAD — an initial counter value
- VAL — a current counter value, decremented on each clock cycle
- CALIB — calibration register

Every time VAL drops to zero, a SysTick interrupt is generated.

The SysTick interrupt index in the vector table is 15, so we need to set it. Upon boot, our STMicroelectronics Nucleo-F429ZI board runs at 16 MHz. We can configure the SysTick counter to trigger an interrupt every millisecond.

First, let's define a SysTick peripheral. We know four registers, and, from the Arm Reference Manual, we see that the SysTick address is 0xe000e010. So:

```
struct systick {
  volatile uint32_t CTRL, LOAD, VAL, CALIB;
};
#define SYSTICK ((struct systick *) 0xe000e010)
```

Next, add an API function that configures it. We need to enable SysTick in the `SYSTICK->CTRL` register, and also clock it via `RCC->APB2ENR`, as described in section 7.3.14 of the manual [4]:

```
#define BIT(x) (1UL << (x))
static inline void systick_init(uint32_t ticks) {
  // SysTick timer is 24 bits
  if ((ticks - 1) > 0xffffff) return;
  SYSTICK->LOAD = ticks - 1;
  SYSTICK->VAL = 0;
  // Enable systick
  SYSTICK->CTRL = BIT(0) | BIT(1) | BIT(2);
  RCC->APB2ENR |= BIT(14); // Enable SYSCFG
}
```

As mentioned, the Nucleo-F429ZI board runs at 16 MHz, which means that if we call `systick_init(16000000 / 1000)`, then a SysTick interrupt will be generated every millisecond. We should have an interrupt handler function defined — here's one that simply increments a 32-bit millisecond counter:

```
// "volatile" is important!!
static volatile uint32_t s_ticks;
void SysTick_Handler(void) {
  s_ticks++;
}
```

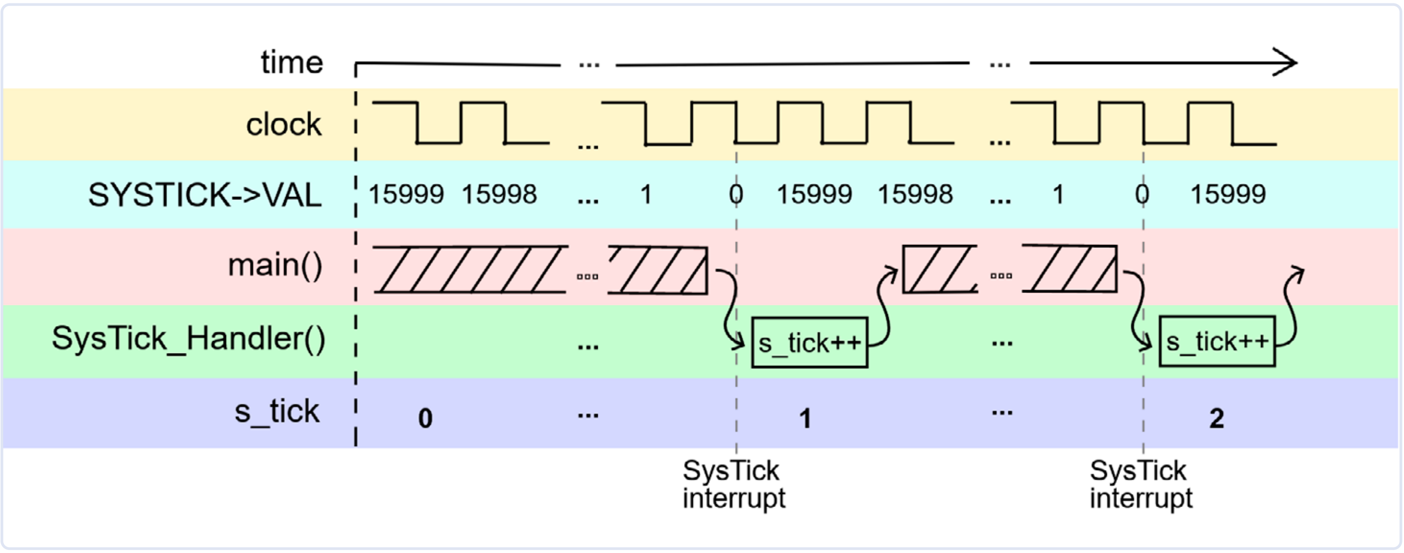


Figure 1: Time scale representation of interrupted firmware execution with the SysTick_Handler() function.

With a 16 MHz clock, we initialize the SysTick counter to trigger an interrupt every 16,000 cycles: The `SYSTICK->VAL` initial value is 15,999, then it decrements every cycle until it reaches 0, when an interrupt is triggered. The firmware code execution gets interrupted, and the `SysTick_Handler()` function is called to increment the `s_tick` variable. **Figure 1** shows how it looks on a timescale.

The `volatile` qualifier is required here because `s_ticks` is modified by the interrupt handler. `volatile` prevents the compiler from optimizing/caching the `s_ticks` value in a CPU register; instead, generated code always accesses memory. That is why the `volatile` qualifier is present in the peripheral `struct` definitions, too. Since this is important to understand, let's demonstrate that with a simple function: Arduino's `delay()`. Let us use our `s_ticks` variable:

```
// This function waits "ms" milliseconds
void delay(unsigned ms) {
  // Time in the future when we need to stop
  uint32_t until = s_ticks + ms; /
  while (s_ticks < until) (void) 0; // Loop until then
}
```

Now let's compile this code both with and without the `volatile` qualifier for `s_ticks` and compare the compiled assembly language code:

```
// NO VOLATILE: uint32_t s_ticks;
ldr r3, [pc, #8] // cache s_ticks
ldr r3, [r3, #0] // in r3
adds r0, r3, r0 // r0 = r3 + ms
cmp r3, r0 // ALWAYS FALSE
bcc.n 200000d2
bx lr

// VOLATILE: volatile uint32_t s_ticks;
ldr r2, [pc, #12]
ldr r3, [r2, #0] // r3 = s_ticks
adds r3, r3, r0  // r3 = r3 + ms
ldr r1, [r2, #0] // RELOAD: r1 = s_ticks
cmp r1, r3 // compare
bcc.n 200000d2
bx lr
```

If there is no `volatile`, the `delay()` function will loop forever and never return. That's because it caches (optimizes) the value of `s_ticks` in a register and never updates it. A compiler does that because it doesn't know that `s_ticks` will be updated elsewhere by the interrupt handler! The code compiled using `volatile`, on the other hand, loads the `s_ticks` value on each iteration. So, the rule of thumb: **Values in memory that get updated by interrupt handlers or by the hardware must be declared as** `volatile`.

Now we should add the `SysTick_Handler()` interrupt handler to the vector table:

```
__attribute__((section(".vectors")))
    void (*tab[16 + 91])(void) = {
    _estack, _reset, 0, 0, 0, 0, 0, 0, 0, 0,
                     0, 0, 0, 0,  SysTick_Handler
};
```

Now we have a precise millisecond clock! Let's create a helper function for arbitrary periodic timers:

```
// t: expiration time, prd: period,
//    now: current time. Return true if expired
bool timer_expired(uint32_t *t, uint32_t prd,
                   uint32_t now) {
  if (now + prd < *t) *t = 0;
    // Time wrapped? Reset timer
  if (*t == 0) *t = now + prd;
    // First poll? Set expiration
  if (*t > now) return false;
    // Not expired yet,return
  *t = (now - *t) > prd ? now + prd : *t + prd;
    // Next expiration time
  return true; // Expired, return true
}
```

Now we're ready to update our main loop and use a precise timer for LED blinking. For example, let's use a 250-millisecond blinking interval:

```
// Declare timer and 500ms period
uint32_t timer, period = 500;
for (;;) {
  if (timer_expired(&timer, period, s_ticks)) {
    static bool on;       // This block is executed
    gpio_write(led, on); // Every "period" milliseconds
    on = !on; // Toggle LED state
  }
  // Here we could perform other activities!
}
```

Note that by using SysTick with a helper `timer_expired()` function, we made our main loop (also called the "superloop") non-blocking. That means that inside that loop we can perform many actions — for example, have different timers with different periods, and they all will be triggered on time.

You can find the complete project source code in the *step-2-systick* folder [5].

Add UART Debug Output

Now it's time to add a human-readable diagnostics to our firmware. One of the MCU peripherals is a serial UART interface. Looking at the memory map in section 2.3 of the microcontroller manual, we see that there are several UART/USART controllers — i.e. pieces of circuitry inside the MCU that, properly configured, can exchange data via certain pins. A minimal UART setup uses two pins: RX (receive) and TX (transmit).

In section 6.9 of the Nucleo board manual [6], we see that one of the controllers, USART3, uses pins PD8 (TX) and PD9 (RX) and is connected to the on-board ST-LINK debugger. That means that if we configure USART3 and output data via the PD9 pin, we can see it on our workstation via the ST-LINK USB connection.

So, let us create a handy API for the UART, like we did for GPIO. Section 30.6 [4] summarizes the UART registers, and here is our corresponding UART `struct`:

```
struct uart {
  volatile uint32_t SR, DR, BRR, CR1, CR2, CR3, GTPR;
};
#define UART1 ((struct uart *) 0x40011000)
#define UART2 ((struct uart *) 0x40004400)
#define UART3 ((struct uart *) 0x40004800)
```

To configure a UART, we need to:

- Enable the UART clock by setting the appropriate bit in `RCC->APB2ENR`.
- Set *alternate function* pin mode for RX and TX pins. There can be several alternate functions (AF) for any given pin, depending on the peripheral that is used. The AF list can be found in the STM32F429ZI datasheet, Table 12 [7].
- Set the baud rate (receive/transmit clock frequency) via the BRR register.
- Enable the peripheral receive and transmit via the CR1 register.

We already know how to set a GPIO pin to a specific mode. If a pin is in the AF mode, we also need to specify the "function number," i.e. which exact peripheral takes control. This can be done via the *Alternate Function Register*, `AFR`, of the GPIO peripheral. Reading the AFR register description in the Reference Manual, we see that the AF number occupies four bits, so the whole setup for 16 pins occupies two registers.

```
static inline void gpio_set_af(uint16_t pin,
                               uint8_t af_num) {
  struct gpio *gpio = GPIO(PINBANK(pin)); // GPIO bank
  int n = PINNO(pin); // Pin number
  gpio->AFR[n >> 3] &= ~(15UL << ((n & 7) * 4));
  gpio->AFR[n >> 3] |= ((uint32_t) af_num)
                      << ((n & 7) * 4);
}
```

In order to completely hide register-specific code from the GPIO API, let's move the GPIO clock init to the `gpio_set_mode()` function:

```
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
// Enable GPIO clock
   RCC->AHB1ENR |= BIT(PINBANK(pin));
   ...
```

Now we're ready to create a UART initialization API function — see **Listing 1**.

Finally, we need functions for reading and writing to the UART. Reference Manual [4] section 30.6.1 tells us that the Status Register, SR, shows whether data is ready:

```
static inline int uart_read_ready(struct uart *uart) {
  // If RXNE bit is set, data is ready
  return uart->SR & BIT(5);
}
```

The data byte itself can be fetched from the Data Register, DR:

```
static inline uint8_t uart_read_byte(struct uart *uart) {
  return (uint8_t) (uart->DR & 255);
}
```

Listing 1. UART initialization API function.

```
#define FREQ 16000000  // CPU frequency, 16 Mhz
 static inline void uart_init(struct uart *uart, unsigned long baud) {
   // https://www.st.com/resource/en/datasheet/stm32f429zi.pdf
   uint8_t af = 7;           // Alternate function
   uint16_t rx = 0, tx = 0;  // pins

   if (uart == UART1) RCC->APB2ENR |= BIT(4);
   if (uart == UART2) RCC->APB1ENR |= BIT(17);
   if (uart == UART3) RCC->APB1ENR |= BIT(18);
   if (uart == UART1) tx = PIN('A', 9), rx = PIN('A', 10);
   if (uart == UART2) tx = PIN('A', 2), rx = PIN('A', 3);
   if (uart == UART3) tx = PIN('D', 8), rx = PIN('D', 9);

  gpio_set_mode(tx, GPIO_MODE_AF);
   gpio_set_af(tx, af);
   gpio_set_mode(rx, GPIO_MODE_AF);
   gpio_set_af(rx, af);
   uart->CR1 = 0;                          // Disable this UART
   uart->BRR = FREQ / baud;                // FREQ is a UART bus frequency
   uart->CR1 |= BIT(13) | BIT(2) | BIT(3);  // Set UE, RE, TE
 }
```

Transmitting a single byte can also be done via the Data register. After setting a byte to write, we need to wait for the transmission to end, which will be indicated via Bit 7 in the Status register:

```
static inline void uart_write_byte(struct uart *uart,
                                   uint8_t byte) {
  uart->DR = byte;
  while ((uart->SR & BIT(7)) == 0) spin(1);
}
```

And writing a buffer:

```
static inline void
  uart_write_buf(struct uart *uart,
                 char *buf, size_t len) {
    while (len-- > 0)
      uart_write_byte(uart, *(uint8_t *) buf++);
}
```

Now, initialize the UART in our `main()` function:

```
…
uart_init(UART3, 115200); // Initialize UART
```

Now, we're ready to print the message, "hi\r\n" every time the LED blinks!

```
if (timer_expired(&timer, period, s_ticks)) {
  …
  uart_write_buf(UART3, "hi\r\n", 4); // Write message
}
```

Rebuild, reflash, and attach a terminal program to the ST-LINK port. On my Mac workstation, I use *cu*. It also can be used on Linux. In Windows, using the PuTTY [8] utility works well. Run a terminal and observe the messages:

```
$ cu -l /dev/cu.YOUR_SERIAL_PORT -s 115200
hi
hi
```

The complete project source code can be found in the *step-3-uart* folder [9].

Redirect printf() to UART

In this section, we replace the `uart_write_buf()` call with a `printf()` call, which gives us the ability to do formatted output — and increase our flexibility in printing diagnostic information, by implementing so-called "printf()-style debugging".

The GNU ARM toolchain that we're using comes not only with a GCC compiler and other tools, but with a C library called *newlib* [10]. The *newlib* library was developed by RedHat for embedded systems.

If our firmware calls a standard C library function, for example `strcmp()`, then a *newlib* code will be added to our firmware by the GCC linker.

Some of the standard C functions that *newlib* implements, specifically file input/output (IO) operations, are implemented by *newlib* in a special fashion: Those functions eventually call a set of low-level IO functions called *syscalls*.

Listing 2. The main() function becomes quite compact.

```
#include "hal.h"

static volatile uint32_t s_ticks;
 void SysTick_Handler(void) {
   s_ticks++;
 }

int main(void) {
   uint16_t led = PIN('B', 7);            // Blue LED
   systick_init(16000000 / 1000);         // Tick every 1 ms
   gpio_set_mode(led, GPIO_MODE_OUTPUT);  // Set blue LED to output mode
   uart_init(UART3, 115200);              // Initialise UART
   uint32_t timer = 0, period = 500;      // Declare timer and 500ms period
   for (;;) {
     if (timer_expired(&timer, period, s_ticks)) {
       static bool on;                     // This block is executed
       gpio_write(led, on);                // Every 'period' milliseconds
       on = !on;                           // Toggle LED state
       uart_write_buf(UART3, "hi\r\n", 4);  // Write message
     }
     // Here we could perform other activities!
   }
   return 0;
 }
```

For example:

- `fopen()` eventually calls `_open()`
- `fread()` eventually calls a low level `_read()`
- `fwrite()`, `fprintf()`, `printf()` eventually call a low level `_write()`
- `malloc()` eventually calls `_sbrk()`, and so on.

Thus, by modifying a `_write()` syscall, we can redirect `printf()` to whatever we want. That mechanism is called "IO retargeting."

Note: The STM32 Cube IDE also uses ARM GCC with *newlib*, that's why Cube projects typically include a *syscalls.c* file. Other toolchains, like TI's CCS and Keil's CC might use a different C library with a slightly different retargeting mechanism. We use *newlib*, so let's modify `_write()` syscall to print to UART3.

Before that, let's organize our source code in the following way:

- Move all API definitions to *mcu.h*
- Move startup code to *startup.c*
- Create an empty file *syscalls.c* for *newlib* syscalls
- Modify the Makefile to add *syscalls.c* and *startup.c* to the build

After moving all API definitions to *mcu.h*, our *main.c* file becomes quite compact. Note that it does not have any mention of the low-level registers, just high-level API functions that are easy to understand — see **Listing 2**.

Great, now let's retarget `printf()` to UART3. In the empty *syscalls.c*, copy/paste the following code:

```
#include "mcu.h"
int _write(int fd, char *ptr, int len) {
  (void) fd, (void) ptr, (void) len;
  if (fd == 1) uart_write_buf(UART3, ptr, (size_t) len);
  return -1;
}
```

Here, we say: If the file descriptor that we're writing to, `fd`, is 1 (which is a standard output descriptor), then write the buffer to UART3. Otherwise, ignore. This is the essence of retargeting!

Rebuilding this firmware results in a bunch of linker errors, as shown in **Listing 3**.

Since we've used a *newlib stdio* function, we need to supply *newlib* with the rest of the syscalls. Let's add just a simple stub that does nothing (**Listing 4**).

Now, a rebuild gives no errors. Last step: replace the `uart_write_buf()` call in the `main()` function with `printf()` call that prints something useful, e.g. an LED status and the current value of systick:

```
// Write message
printf("LED: %d, tick: %lu\r\n", on, s_ticks);
```

Listing 3. Bunch of linker errors.

```
../../arm-none-eabi/lib/thumb/v7e-m+fp/hard/libc_nano.a(lib_a-sbrkr.o): in function `_sbrk_r':
 sbrkr.c:(.text._sbrk_r+0xc): undefined reference to `_sbrk'
 closer.c:(.text._close_r+0xc): undefined reference to `_close'
 lseekr.c:(.text._lseek_r+0x10): undefined reference to `_lseek'
 readr.c:(.text._read_r+0x10): undefined reference to `_read'
 fstatr.c:(.text._fstat_r+0xe): undefined reference to `_fstat'
 isattyr.c:(.text._isatty_r+0xc): undefined reference to `_isatty'
```

Listing 4. Adding simple stubs.

```
int _fstat(int fd, struct stat *st) {
   (void) fd, (void) st;
   return -1;
 }

void *_sbrk(int incr) {
   (void) incr;
   return NULL;
 }

int _close(int fd) {
   (void) fd;
   return -1;
 }

int _isatty(int fd) {
   (void) fd;
   return 1;
 }

int _read(int fd, char *ptr, int len) {
   (void) fd, (void) ptr, (void) len;
   return -1;
 }

int _lseek(int fd, int ptr, int dir) {
   (void) fd, (void) ptr, (void) dir;
   return 0;
 }
```

The serial output looks like this:

```
LED: 1, tick: 250
LED: 0, tick: 500
LED: 1, tick: 750
LED: 0, tick: 1000
```

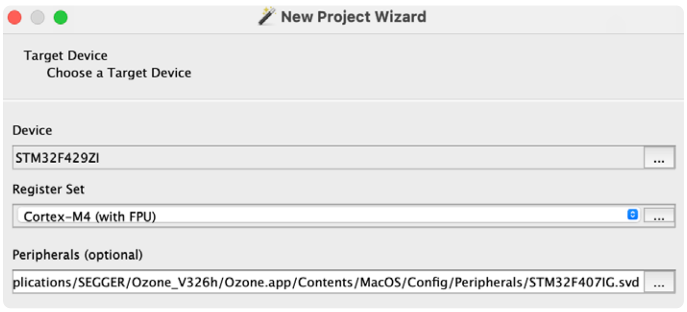
Congratulations! We learned how IO retargeting works, and can now printf()-debug our firmware. You can find the complete project source code the *step-4-printf* folder [11].

Debug with Segger Ozone

What if our firmware is stuck somewhere and `printf()` debug does not work? What if even the startup code does not work? We would need a debugger. There are many options, but I'd recommend using the Ozone debugger from Segger. Why? Because it's standalone; it does not need any IDE set up. We can feed our *firmware.elf* file directly to Ozone, and it'll pick up our source files.

So, download Ozone from the Segger website [12]. Before we can use it with our Nucleo board, we need to convert the ST-LINK firmware on the onboard debugger to the *jlink* firmware that Ozone understands. Follow the instructions on the Segger site [13], then:

- Run Ozone. Choose our device in the wizard (**Figure 2**).
- Select a debugger we're going to use — that should be an ST-LINK (**Figure 3**).


New Project Wizard

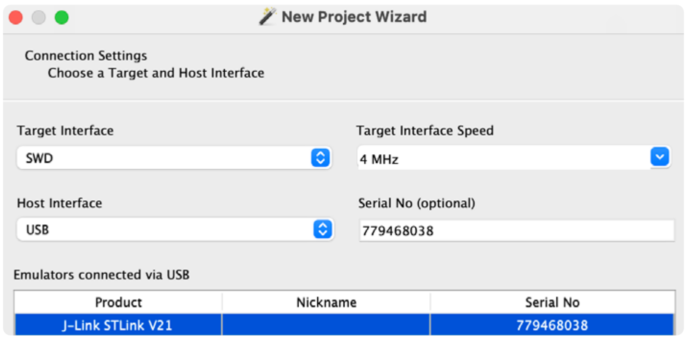
Target Device
Choose a Target Device

Device: STM32F429ZI

Register Set: Cortex-M4 (with FPU)

Peripherals (optional): plications/SEGGER/Ozone_V326h/Ozone.app/Contents/MacOS/Config/Peripherals/STM32F407IG.svd

Figure 2: Select the device in the wizard.


New Project Wizard

Connection Settings
Choose a Target and Host Interface

Target Interface: SWD

Target Interface Speed: 4 MHz

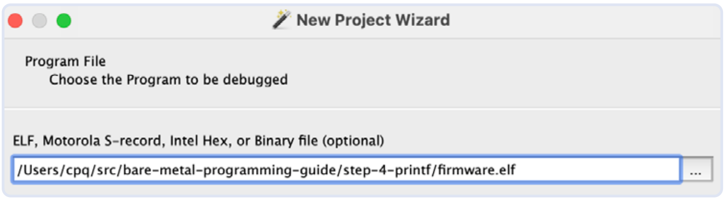
Host Interface: USB

Serial No (optional): 779468038

Emulators connected via USB

Product	Nickname	Serial No
J-Link STLink V21		779468038

Figure 3: Select STLink as debugger.



New Project Wizard

Program File
Choose the Program to be debugged

ELF, Motorola S-record, Intel Hex, or Binary file (optional)
/Users/cpq/src/bare-metal-programming-guide/step-4-printf/firmware.elf

Figure 4: The program to be debugged will be our firmware.elf file.

- Choose our *firmware.elf* file (**Figure 4**).
- Leave the defaults on the next screen, click *Finish*, and we've got our debugger loaded (note the *mcu.h* source code is picked up), see **Figure 5**.
- Click the green button to download, run the firmware, and we're stopped here (**Figure 6**).

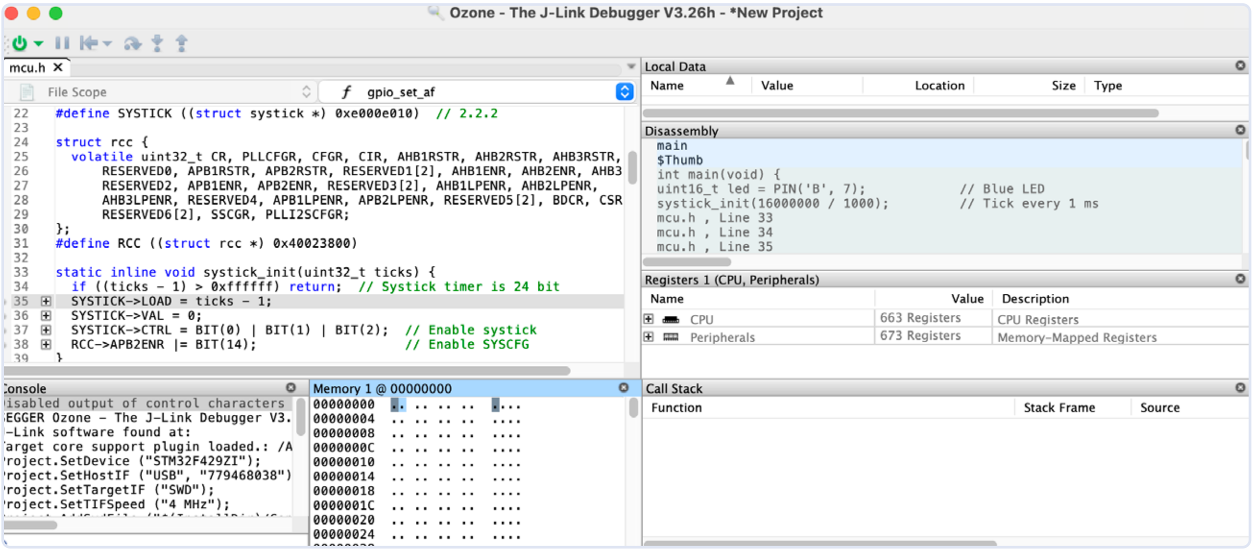
Now we can single-step through code, set breakpoints, and do the usual debugging stuff. One thing that should be noted is the handy Ozone *Peripherals* view (**Figure 7**). Using it, we can directly examine or set the state of the peripherals. For example, let's turn on a green on-board LED (PB0):

1. We need to clock GPIOB first. Find *Peripherals* → *RCC* → *AHB1ENR*, and enable the *GPIOBEN* bit — set it to 1 (**Figure 8**).
2. Find *Peripherals* → *GPIO* → *GPIOB* → *MODER*, set *MODER0* to 1 (output) (**Figure 9**).
3. Find *Peripherals* → *GPIO* → *GPIOB* → *ODR*, set *ODR0* to 1 (on) (**Figure 10**).

Now, a green LED should be on! Happy debugging!

In the third part of this series, we will implement a web server. Furthermore, we will show how a program can be tested automatically, and much more. Stay tuned!

220665-B-01



Ozone - The J-Link Debugger V3.26h - *New Project

mcu.h
File Scope | gpio_set_af

```
#define SYSTICK ((struct systick *) 0xe000e010)  // 2.2.2

struct rcc {
  volatile uint32_t CR, PLLCFGR, CFGR, CIR, AHB1RSTR, AHB2RSTR, AHB3RSTR,
      RESERVED0, APB1RSTR, APB2RSTR, RESERVED1[2], AHB1ENR, AHB2ENR, AHB3
      RESERVED2, APB1ENR, APB2ENR, RESERVED3[2], AHB1LPENR, AHB2LPENR,
      AHB3LPENR, RESERVED4, APB1LPENR, APB2LPENR, RESERVED5[2], BDCR, CSR
      RESERVED6[2], SSCGR, PLLI2SCFGR;
};
#define RCC ((struct rcc *) 0x40023800)

static inline void systick_init(uint32_t ticks) {
  if ((ticks - 1) > 0xffffff) return;  // Systick timer is 24 bit
  SYSTICK->LOAD = ticks - 1;
  SYSTICK->VAL = 0;
  SYSTICK->CTRL = BIT(0) | BIT(1) | BIT(2);  // Enable systick
  RCC->APB2ENR |= BIT(14);                   // Enable SYSCFG
}
```

Local Data

Name	Value	Location	Size	Type

Disassembly

```
main
$Thumb
int main(void) {
uint16_t led = PIN('B', 7);          // Blue LED
systick_init(16000000 / 1000);       // Tick every 1 ms
mcu.h , Line 33
mcu.h , Line 34
mcu.h , Line 35
```

Registers 1 (CPU, Peripherals)

Name	Value	Description
CPU	663 Registers	CPU Registers
Peripherals	673 Registers	Memory-Mapped Registers

Console

```
isabled output of control characters
EGGER Ozone - The J-Link Debugger V3.
-Link software found at:
arget core support plugin loaded.: /A
roject.SetDevice ("STM32F429ZI");
roject.SetHostIF ("USB", "779468038")
roject.SetTargetIF ("SWD");
roject.SetTIFSpeed ("4 MHz");
```

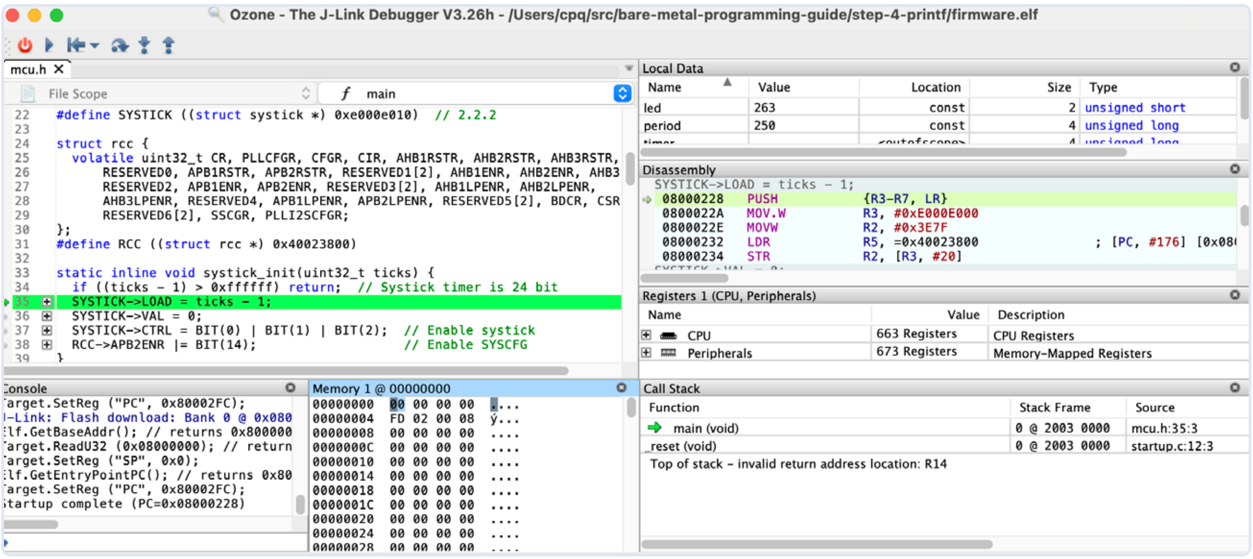
Memory 1 @ 00000000

```
00000000  .. .. .. ..  ....
00000004  .. .. .. ..  ....
00000008  .. .. .. ..  ....
0000000C  .. .. .. ..  ....
00000010  .. .. .. ..  ....
00000014  .. .. .. ..  ....
00000018  .. .. .. ..  ....
0000001C  .. .. .. ..  ....
00000020  .. .. .. ..  ....
00000024  .. .. .. ..  ....
```

Call Stack

Function	Stack Frame	Source

Figure 5: The debugger is loaded, and soon mcu.h appears.



Ozone - The J-Link Debugger V3.26h - /Users/cpq/src/bare-metal-programming-guide/step-4-printf/firmware.elf

mcu.h
File Scope | main

```
#define SYSTICK ((struct systick *) 0xe000e010)  // 2.2.2

struct rcc {
  volatile uint32_t CR, PLLCFGR, CFGR, CIR, AHB1RSTR, AHB2RSTR, AHB3RSTR,
      RESERVED0, APB1RSTR, APB2RSTR, RESERVED1[2], AHB1ENR, AHB2ENR, AHB3
      RESERVED2, APB1ENR, APB2ENR, RESERVED3[2], AHB1LPENR, AHB2LPENR,
      AHB3LPENR, RESERVED4, APB1LPENR, APB2LPENR, RESERVED5[2], BDCR, CSR
      RESERVED6[2], SSCGR, PLLI2SCFGR;
};
#define RCC ((struct rcc *) 0x40023800)

static inline void systick_init(uint32_t ticks) {
  if ((ticks - 1) > 0xffffff) return;  // Systick timer is 24 bit
  SYSTICK->LOAD = ticks - 1;
  SYSTICK->VAL = 0;
  SYSTICK->CTRL = BIT(0) | BIT(1) | BIT(2);  // Enable systick
  RCC->APB2ENR |= BIT(14);                   // Enable SYSCFG
}
```

Local Data

Name	Value	Location	Size	Type
led	263	const	2	unsigned short
period	250	const	4	unsigned long

Disassembly

```
SYSTICK->LOAD = ticks - 1;
08000228   PUSH      {R3-R7, LR}
0800022A   MOV.W     R3, #0xE000E000
0800022E   MOVW      R2, #0x3E7F
08000232   LDR       R5, =0x40023800            ; [PC, #176] [0x08
08000234   STR       R2, [R3, #20]
```

Registers 1 (CPU, Peripherals)

Name	Value	Description
CPU	663 Registers	CPU Registers
Peripherals	673 Registers	Memory-Mapped Registers

Console

```
arget.SetReg ("PC", 0x80002FC);
-Link: Flash download: Bank 0 @ 0x080
lf.GetBaseAddr(); // returns 0x800000
arget.ReadU32 (0x08000000); // return
arget.SetReg ("SP", 0x0);
lf.GetEntryPointPC(); // returns 0x80
arget.SetReg ("PC", 0x80002FC);
tartup complete (PC=0x08000228)
```

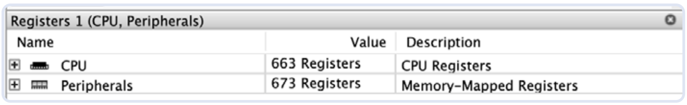
Memory 1 @ 00000000

```
00000000  00 00 00 00  ....
00000004  FD 02 00 08  ý...
00000008  00 00 00 00  ....
0000000C  00 00 00 00  ....
00000010  00 00 00 00  ....
00000014  00 00 00 00  ....
00000018  00 00 00 00  ....
0000001C  00 00 00 00  ....
00000020  00 00 00 00  ....
00000024  00 00 00 00  ....
00000028  00 00 00 00  ....
```

Call Stack

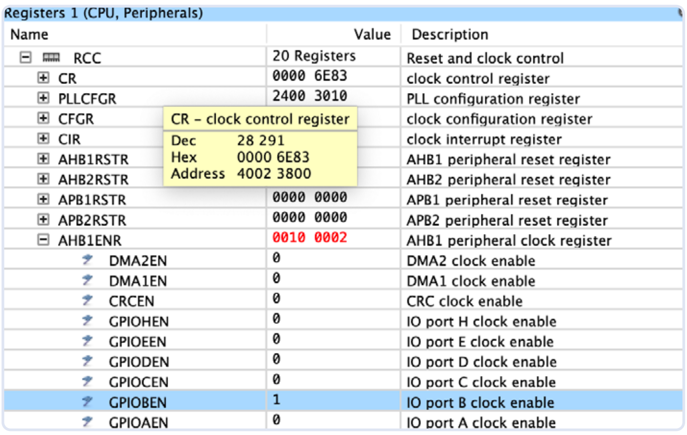
Function	Stack Frame	Source
main (void)	0 @ 2003 0000	mcu.h:35:3
_reset (void)	0 @ 2003 0000	startup.c:12:3
Top of stack - invalid return address location: R14		

Figure 6: After we run the firmware, it halts on the SYSTICK->LOAD = ticks - 1; line.


Registers 1 (CPU, Peripherals)

Name	Value	Description
⊞ CPU	663 Registers	CPU Registers
⊞ Peripherals	673 Registers	Memory-Mapped Registers

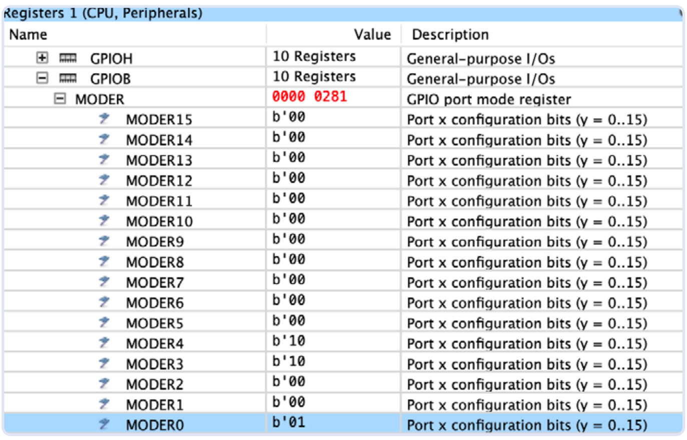
Figure 7: Convenient Ozone Peripherals view for easy examination and configuration of peripherals.


Registers 1 (CPU, Peripherals)

Name	Value	Description
⊟ RCC	20 Registers	Reset and clock control
⊞ CR	0000 6E83	clock control register
⊞ PLLCFGR	2400 3010	PLL configuration register
⊞ CFGR		clock configuration register
⊞ CIR		clock interrupt register
⊞ AHB1RSTR		AHB1 peripheral reset register
⊞ AHB2RSTR		AHB2 peripheral reset register
⊞ APB1RSTR	0000 0000	APB1 peripheral reset register
⊞ APB2RSTR	0000 0000	APB2 peripheral reset register
⊟ AHB1ENR	0010 0002	AHB1 peripheral clock register
DMA2EN	0	DMA2 clock enable
DMA1EN	0	DMA1 clock enable
CRCEN	0	CRC clock enable
GPIOHEN	0	IO port H clock enable
GPIOEEN	0	IO port E clock enable
GPIODEN	0	IO port D clock enable
GPIOCEN	0	IO port C clock enable
GPIOBEN	1	IO port B clock enable
GPIOAEN	0	IO port A clock enable

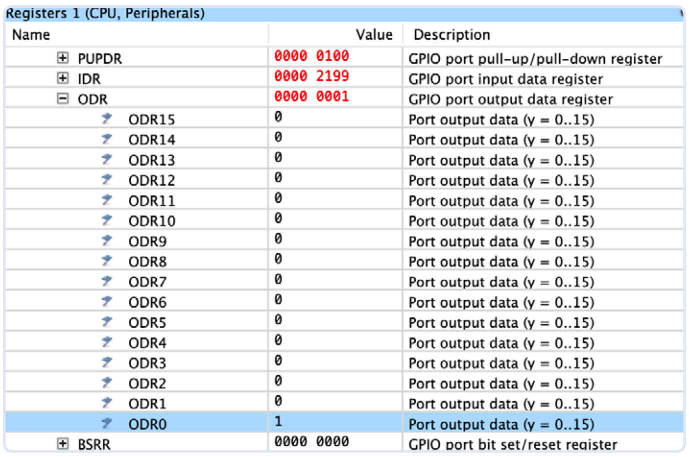
CR – clock control register
Dec 28 291
Hex 0000 6E83
Address 4002 3800

Figure 8: Enabling the running of the clock on Port B by setting the value of GPIOBEN to 1.


Registers 1 (CPU, Peripherals)

Name	Value	Description
⊞ GPIOH	10 Registers	General-purpose I/Os
⊟ GPIOB	10 Registers	General-purpose I/Os
⊟ MODER	0000 0281	GPIO port mode register
MODER15	b'00	Port x configuration bits (y = 0..15)
MODER14	b'00	Port x configuration bits (y = 0..15)
MODER13	b'00	Port x configuration bits (y = 0..15)
MODER12	b'00	Port x configuration bits (y = 0..15)
MODER11	b'00	Port x configuration bits (y = 0..15)
MODER10	b'00	Port x configuration bits (y = 0..15)
MODER9	b'00	Port x configuration bits (y = 0..15)
MODER8	b'00	Port x configuration bits (y = 0..15)
MODER7	b'00	Port x configuration bits (y = 0..15)
MODER6	b'00	Port x configuration bits (y = 0..15)
MODER5	b'00	Port x configuration bits (y = 0..15)
MODER4	b'10	Port x configuration bits (y = 0..15)
MODER3	b'10	Port x configuration bits (y = 0..15)
MODER2	b'00	Port x configuration bits (y = 0..15)
MODER1	b'00	Port x configuration bits (y = 0..15)
MODER0	b'01	Port x configuration bits (y = 0..15)

Figure 9: Set MODER0 to 1 (and thus, selecting output) in the GPIO peripherals.


Registers 1 (CPU, Peripherals)

Name	Value	Description
⊞ PUPDR	0000 0100	GPIO port pull-up/pull-down register
⊞ IDR	0000 2199	GPIO port input data register
⊟ ODR	0000 0001	GPIO port output data register
ODR15	0	Port output data (y = 0..15)
ODR14	0	Port output data (y = 0..15)
ODR13	0	Port output data (y = 0..15)
ODR12	0	Port output data (y = 0..15)
ODR11	0	Port output data (y = 0..15)
ODR10	0	Port output data (y = 0..15)
ODR9	0	Port output data (y = 0..15)
ODR8	0	Port output data (y = 0..15)
ODR7	0	Port output data (y = 0..15)
ODR6	0	Port output data (y = 0..15)
ODR5	0	Port output data (y = 0..15)
ODR4	0	Port output data (y = 0..15)
ODR3	0	Port output data (y = 0..15)
ODR2	0	Port output data (y = 0..15)
ODR1	0	Port output data (y = 0..15)
ODR0	1	Port output data (y = 0..15)
⊞ BSRR	0000 0000	GPIO port bit set/reset register

Figure 10: Turning ODR0 on by selecting value 1 in ODR (GPIO).

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.

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.

WEB LINKS

[1] GitHub Repository for this article: https://github.com/cpq/bare-metal-programming-guide
[2] Sergey Lyubka, "A Bare-Metal Programming Guide (Part 1)," Elektor 9-10/2023: https://elektormagazine.com/220665-01
[3] Arm v7-M Architecture Reference Manual: https://developer.arm.com/documentation/ddi0403/ee
[4] Reference Manual RM0090 for STM32F429: https://bit.ly/3neE7S7
[5] Step 2 SysTick folder: https://github.com/cpq/bare-metal-programming-guide/tree/main/steps/step-2-systick
[6] Nucleo-144 Board User Manual (UM1974): https://bit.ly/3oIBXKZ
[7] STM32F429ZI Datasheet: https://st.com/resource/en/datasheet/stm32f429zi.pdf
[8] PuTTY: https://putty.org
[9] Step 3 UART folder: https://github.com/cpq/bare-metal-programming-guide/tree/main/steps/step-3-uart
[10] newlib C library : https://sourceware.org/newlib
[11] Step 4 printf Folder: https://github.com/cpq/bare-metal-programming-guide/tree/main/steps/step-4-printf
[12] Ozone — The J-Link Debugger and Performance Analyzer:
https://segger.com/products/development-tools/ozone-j-link-debugger
[13] Converting ST-LINK On-Board Into a J-Link:
https://segger.com/products/debug-probes/j-link/models/other-j-links/st-link-on-board

elektor store

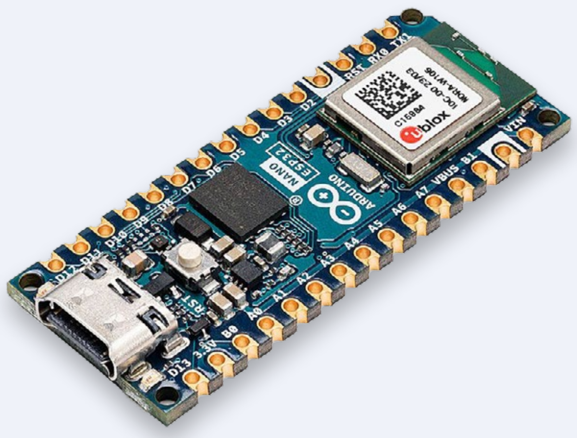
www.elektor.com

The Elektor Store

Never expensive, always surprising

The Elektor Store developed from the community store for Elektor's own products, such as books, magazines, kits and modules, into a mature web store that offers great value for surprising electronics. We offer the products that we ourselves are enthusiastic about or that we simply want to try out. If you have a nice suggestion, we are here: sale@elektor.com.

Arduino Nano ESP32



The Arduino Nano ESP32 (with and without headers) is a Nano form factor board based on the ESP32-S3 (embedded in the NORA-W106-10B from u-blox). This is the first Arduino board to be based fully on an ESP32, and features Wi-Fi, Bluetooth LE, debugging via native USB in the Arduino IDE as well as low power.

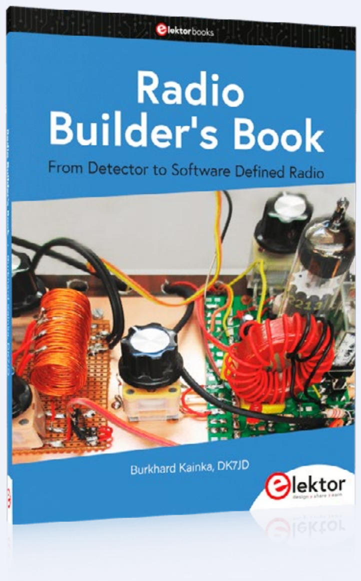
Price: €23.95

Member Price: €21.56

www.elektor.com/20562

Radio Builder's Book

Radio frequency (RF) technology is one of the areas which still allows putting your own ideas into practice. Countless circuit variants with special objectives allow space for meaningful experiments and projects. Many things simply aren't available off the shelf.



Price: €34.95

Member Price: €31.46

www.elektor.com/20553

Raspberry Pi 4 (4 GB) Official Starter Kit

Price: €104.95

www.elektor.com/20556

QuantAsylum QA403 24-bit Audio Analyzer

QuantAsylum
QA403 24-bit Audio Analyzer
Features
• 24-bit ADC/DAC
• Up to 192KSPS
• Fully Isolated from PC
• Differential Input/Output
• USB Powered
• Built-in Attenuator
Attention
• Do not use this equipment for measuring potentially hazardous or lethal voltages.
• The QA403 current can exceed USB capabilities. See the manual for solutions.
• Maximum Inputs:
REST Interface: HTTP://localhost:9403
CAT I
OUTPUT
+32 dBV Max
INPUT

Price: €799.00

Member Price: €719.10

www.elektor.com/20530

FNIRSI DSO-TC3 (3-in-1) Oscilloscope, Component Tester & Signal Generator

Price: ~~€74.95~~

Special Price: €59.95

www.elektor.com/20520

ESP Terminal (ESP32-S3 based Dev Board with 3.5" Display)

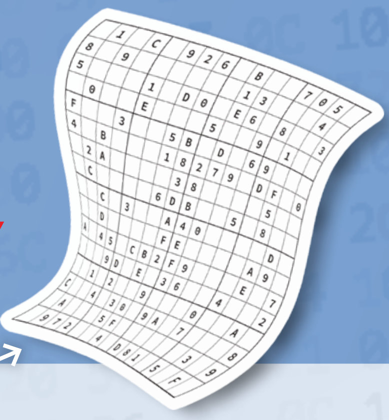
Price: €44.95

Member Price: €40.46

www.elektor.com/20526

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 October 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 to the Hexadoku in our July/August edition is: **23BDF**.
Solutions submitted to us before August 15th were entered in a prize draw for 5 Elektor Store vouchers.
The winners are posted at elektormagazine.com/hexadoku.

Congratulations, everyone!

		E	3	F	B		5	8		2	6	4	0		
B					E	A			5	9					3
4					2	0	9	E	7	B					D
	7								▨	▨	▨	▨	▨	5	
9		B	E	7	4					8	1	3	2		C
		4	6	9		C	A	0	B		7	D	F		
7		A	D	8		B			4		2	5	6		1
5	8			D	0	2			3	6	A			B	4
6	3			B	5	7			E	A	C			8	0
8		0	1	E		3			6		5	9	B		A
		D	B	4		6	2	3	F		9	C	7		
C		2	7	0	A					4	D	F	3		5
	9													7	
E					9	4	3	5	2	C					6
0					7	F			A	D					9
		8	4	5	C		B	7		F	0	A	D		

D	8	1	6	0	E	2	7	4	A	5	9	3	F	C	B
E	3	4	C	F	A	B	9	0	1	8	2	7	6	D	5
5	2	F	9	1	C	6	4	3	B	7	D	8	E	0	A
A	0	B	7	D	3	5	8	E	6	C	F	1	2	4	9
3	B	7	D	4	6	8	1	9	C	F	5	E	0	A	2
F	4	E	1	2	B	A	5	6	D	0	7	9	C	8	3
6	C	2	5	3	7	9	0	A	8	4	E	B	D	F	1
0	9	A	8	C	F	D	E	B	2	1	3	4	5	6	7
7	D	6	A	8	4	F	C	5	E	3	1	2	B	9	0
2	E	8	B	5	9	0	D	7	4	A	C	6	1	3	F
9	F	C	3	E	1	7	6	8	0	2	B	A	4	5	D
1	5	0	4	A	2	3	B	D	F	9	6	C	7	E	8
B	6	D	F	9	8	1	3	C	7	E	0	5	A	2	4
C	1	3	0	B	5	4	2	F	9	6	A	D	8	7	E
4	7	5	2	6	0	E	A	1	3	D	8	F	9	B	C
8	A	9	E	7	D	C	F	2	5	B	4	0	3	1	6

The competition is not open to employees of Elektor International Media, its subsidiaries, licensees and/or associated publishing houses.

PROTEUS DESIGN SUITE

Driving forward with Manual Routing

Push and Shove Routing for dense layouts
Dedicated Differential Pairs Routing mode
Length Matching and Net Tuning Support
Visual DRC shows legal paths for route placement
HEAVY TRAFFIC
FASTER ROUTING AVAILABLE
x 1000
km/h
4WD
H
C
F
E

labcenter Electronics

www.labcenter.com

info@labcenter.com

Imagine if the Wright brothers had partnered with Mouser

With *the widest selection of the newest products,*™ the sky's the limit on what engineers can design.

mouser.co.uk/new

+44 (0) 1494-427500

MOUSER ELECTRONICS