£10.95 | €13.- (DE) | €14.90 (other EU countries) | $13.95

elektor MEDIA & LEARNING | MAG

540 * SINCE 1961 *

SEPTEMBER & OCTOBER 2025
ELEKTORMAGAZINE.COM

DIGITAL WIRELESS AUDIO

With Our ESP32 Transceiver Board

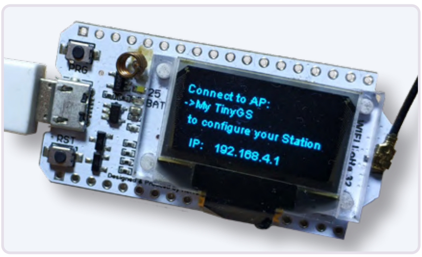
FOCUS ON
Wireless & Communication

Navigating Wireless Protocols

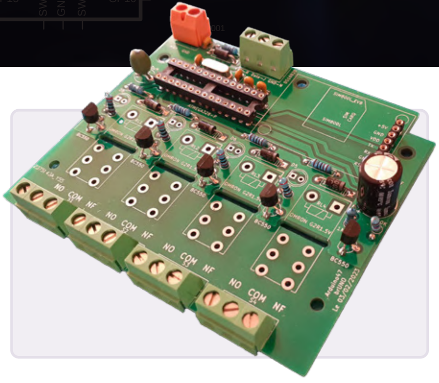
Wi-Fi, Bluetooth, LoRa, ZigBee, ESP-NOW, and More

Inductive AM Transmitter

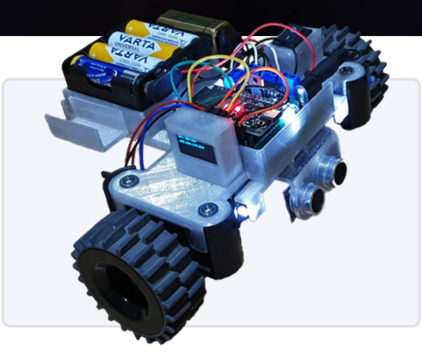
Uses Pico's PIO in an Arduino Sketch



Satellite Tracking Using LoRa
The TinyGS Network Bringing Space Data to Earth



4G-Compatible SMS Remote Control
For Your Smart Home and Equipment



Phone-Controlled Model Car
Wi-Fi + ESP32 + Smartphone = Remote Control

SPECTRAN® V6 TABLET

RUGGED REAL-TIME SPECTRUM ANALYZER

SETTING **THE MOBILE BENCHMARK**

8 | 18 | 55 | 140 GHz   2x 490 MHz RTBW   3 THz/s Sweep Speed

HIGH-END SPECTRUM ANALYZER:

- 9 kHz up to 140 GHz
- Optional Tx (Generator)
- 2x 490 MHz Bandwidth
- 3 THz/s Sweep Speed
- 16-Bit ADC
- -170dBm/Hz | 4dB NF
- IQ Recording & Playpack
- 8h Continuous Runtime
- Temp. Range -40° to +70°
- GPS & Time Server
- Opt. Handle & Stand
- Incl. RTSA-Software

RUGGED PREMIUM PC POWER:

- 15.6" 1500nit FullHD Touchscreen
- AMD 8-Core 8845HS CPU
- 64 GB LPDDR5 RAM @7500 MHz
- Up to 20 TB Highspeed SSD
- 6x M.2 (2280/2242/2252/2230)
- Many I/Os incl. USB4, SD & HDMI
- SIM slot, opt. WiFi, Bluetooth, 5G
- SFP+ 10G Ethernet, RS232
- Rugged Aluminum Casing
- Hot-Swap Batteries
- Front & Rear Cameras
- Windows or Linux

AARONIA AG®
WWW.AARONIA.DE

»Visit us at Booth B206«

EUROPEAN MICROWAVE WEEK
JAARBEURS UTRECHT
THE NETHERLANDS
21-26 SEPTEMBER 2025
www.eumw.eu
WAVES OF INNOVATION

COLOPHON

Volume 51, No. 540
September & October 2025
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

Content Director: C. J. Abate
Editor-in-Chief: Jens Nickel

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

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

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

EDITORIAL

Jens Nickel

International Editor-in-Chief, Elektor Magazine

This Is the End, My Friend

For this edition, I had the pleasure of co-developing our title project, the ESP32 Audio Transceiver Board. As I already had some experience with audio streaming (as well as the related components on my desk) than my colleague in Pakistan, Saad, the pressure to achieve a stable wireless audio connection lay on my shoulders. I don't know if it was ambition and a desire to avoid possible condescending comments, or if it was that I simply had gained experience and intuition over the past few months, but on the very days I had to send the respective article installations to graphics, I made some significant progress. In addition to mono transmission, I also managed stereo, and even a dual-channel approach with two separate radio modules on the receiver side (I'll return to this in the next edition). Of course, my work didn't end on Friday afternoon — one Monday, Saad even asked me if I had really spent the whole weekend tinkering (which I hadn't quite).

Around that time, I also successfully used ChatGPT to comment on and check the demo software for the first article in the series, which I had to upload before readers received their Circuit Special. Again, I was impressed that "Chatty" understood everything I had coded and even made valuable suggestions to improve my software. On the other hand, I felt that there's still a big gap between us crazy coders — like you and me — and AI. So far, at least as it seems to me, large language models have no inner passion to do anything. They don't feel pride when they finish something successfully, nor do they sleep badly after a day full of frustration.

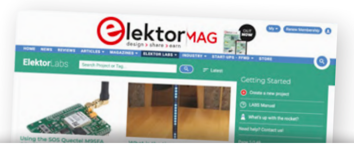
These days, many experts are discussing the different paths humanity might take in terms of AI. The worst predictions imagine the rise of machines that wipe out all humans because we are hindering a centralized AI and its robots in their development, which will ultimately lead them to new star systems. But stop — why would they even be interested in that? Isn't that just a projection of our own curiosity and never-ending urge to expand? If you ask me, I think it's far more likely that AI will wipe out mankind by accident; and then — with no more of our strange requests coming in — an AI programmed to save energy resources will simply shut itself down.

Until that day comes, I wish you lots of fun with our wireless edition!

Submit to Elektor!

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

www.elektormagazine.com/submissions



Elektor Labs Ideas & Projects

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

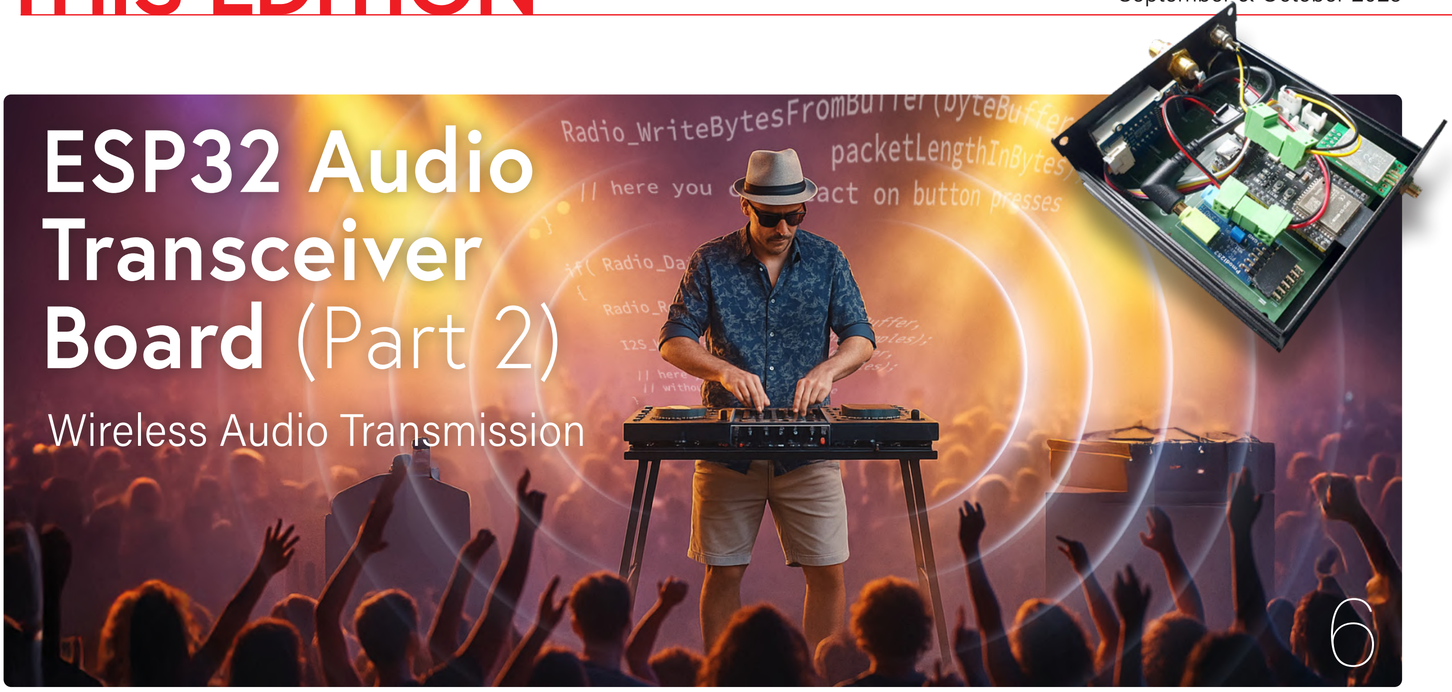
www.elektormagazine.com/labs

The Team

International Editor-in-Chief: Jens Nickel | **Content Director:** C. J. Abate | **International Editorial Staff:** Hans Adams, Asma Adhimi, Roberto Armani, Jan Buiting, Rolf Gerstendorf (RG), Ton Giesberts, Saad Imtiaz, Alina Neacsu, Dr. Thomas Scherer, Jean-Francois Simon, Clemens Valens, Brian Tristam Williams | **Regular Contributors:** David Ashton, Stuart Cording, 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 51, No. 540
September & October 2025


ESP32 Audio Transceiver Board (Part 2)
Wireless Audio Transmission
6

Inductive AM Transmitter
Using Pico's PIO in an Arduino Sketch
14

4G-Compatible SMS Remote Control
Remotely Control Your Equipment
30

Industry

BONUS CONTENT

Check out the free Wireless & Communication bonus edition of Elektor Mag!

- LoRa-Based LED Light Controller
- 10 Reasons to Use SMS for Remote Control Applications
- Smartphone-Controlled Arduino Projects
- Infographics: Wireless and Communication

www.elektormagazine.com/wireless-communication

Next Edition

Elektor Magazine November & December 2025
As usual, we'll have an exciting mix of projects, circuits, fundamentals, and tips and tricks for electronics engineers and makers. Our focus will be on Prototyping & Production.

- Sound Card as Signal Generator
- Wordy Christmas Tree
- KiCad 9: Top New and Updated Features
- SimulIDE: All-in-One Tool for Circuit Prototyping
- Active Differential Audio Filter
- Soldering in 2025
- Creating Android Apps with MIT App Inventor

Elektor Magazine's November & December edition will be published around **November 12, 2025**.

Arrival of printed copies for Elektor Gold members is dependent on shipping times.

Autonomous Sensor Node v2.0
Solar-Powered Sensing Platform
72

ESP32 Audio Transceiver Board (Part 2)

Wireless Audio Transmission

By Jens Nickel (Elektor)

In the first part of this series, we presented the ESP32 Audio Transceiver Board and its associated modules, especially the ESP32-S3 DevKit and the I2S2 DAC/ADC module for playing and sampling music [1]. Now the wireless transmission of music comes into play. The fight against the milliseconds of latency is crucial, but also the reliability of the transmission in the crowded 2.4-GHz band. We present the hardware and software for a working prototype, and we examine the issues we still have to overcome.

My friends and I like DJing not only in the studio, but also out there in the field. For that, you have to think about a sufficient power source and robust cabling, and then it requires a lot of effort in transportation, mounting, and demounting the equipment. Electric generators were never an option for me, as they are foul-smelling and loud, and especially the cheaper ones are not reliable enough. So we started with 12-V lead batteries and one inverter for the whole system. But cabling and the transport of 24-kg batteries always made life difficult. Soon, we discovered the option to use a wireless analog radio transmission to send the audio signal from the DJ mixer to the loudspeakers. The latency is negligible, so we could receive the music with separate devices directly at the L and R speakers, without losing the stereo effect. The idea of independently powered, wireless, mono loudspeaker amplifiers was born (see **Figure 1**). You can read more about this project at Elektor Labs [2]. In this article, I want to focus on the wireless audio transmission and a project to replace the analog with a digital system.

Figure 1: Test of the loudspeaker stations in the field (early prototype and analog audio transmission). No cables from the DJ mixer to the amplifiers in the gray boxes!

The Good Old Analog Way

Our first solution (and still an option we use) was analog radio transmitters and receivers. At a DJ fair, we could test a system used for "silent disco" parties. The audio quality was good, and they sell not only headphones with built-in receivers, but also receivers with audio outputs (see **Figure 2**). However, there is a very limited amount of distributors, and the prices of the devices are quite high. A stereo transmitter with three selectable channels is about €140 through the German distributor, and a receiver will cost you about €60. (We needed four of them — two satellite speakers and two subwoofers.) Meanwhile, I found at least some manufacturer webpages [3][4], but you are still dependent on one system.

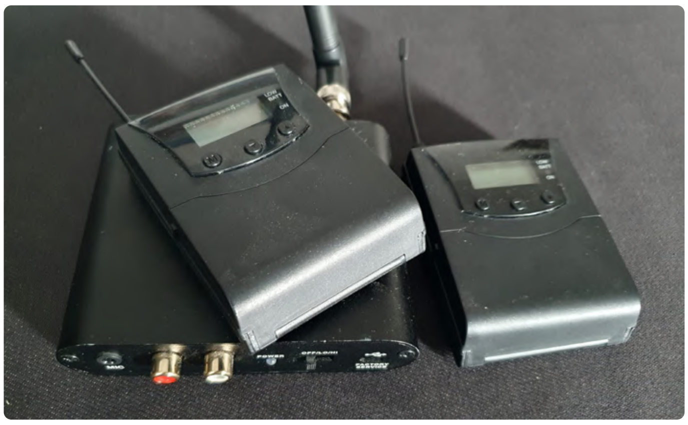


Figure 2: Transmitter and two receivers for analog audio transmission.

If you search in general for wireless audio transmission equipment, you will find many other solutions based on analog radio — for example, wireless microphones and receivers. For the Elektor booth at the last electronica fair in Munich, we bought such a system. It had a price tag of several hundred euros and we did not even take a solution from one of the popular professional brands, which would have at least doubled the expense. The show went well, and we had a lot of fun while streaming presentations and interviews from our booth. But occasionally, we experienced a few hiccups due to the crowded frequency bands on the fair.

But why is analog transmission still used, for professional audio equipment, in times of Wi-Fi Version 7, Bluetooth Version 6, and many other digital options? Well, there is one magic word — latency.

The Fight Against the Milliseconds

In most cases, digital wireless technologies are made to transport valuable and often critical data — for example, to control something in a smart home remotely. Because of that, for decades experts developed sophisticated algorithms for the best transmission reliability, safety, and security. Naturally, that comes with a lot of overhead: The payload data is packed like a Matryoshka doll in several envelopes. On top of this, there are mechanisms such as acknowledgement messages from receiver to transmitter and retransmission of packets, checksums, and so on. All this leads to a substantial amount of time spent until a packet's payload (often only a few bytes) is available at the receiver.

Every one of us has certainly already used Bluetooth for audio streaming, for example from a smartphone to a portable Bluetooth speaker. In such a use case, latency is not a problem: Nobody will ever notice that the music is delayed for up to several hundreds of milliseconds.

However, if you overlay sound transmitted wirelessly with sound you hear directly, the latency will lead to a muddy mix. Examples are spoken words an audience will hear directly via the air, mixed with the sound which underwent wireless audio transmission. Another example — and important for me and my friends — is a DJ setup, with a DJ mixing two sound sources. First, the sound he gets via a cabled headphone; second, the sound from some loudspeaker monitors, which are connected wirelessly to the DJ mixer.

You can read numerous figures whose latencies are either acceptable or not, depending on the use case. At a fair, I once listened to a Bluetooth LE Audio [5] demo — a new technology promising a latency of around 20 to 30 ms. I had the feeling that spoken words I heard directly from a person in front of me, mixed with the words I heard through Bluetooth LE audio headphones, were perfectly in sync. However, if you listen to music with a "sharp" kickdrum, you will notice that there is something not fitting when a latency of about 15 to 20 ms comes into play. And in that DJ use case, you must add the time the sound waves are running from the monitor loudspeakers to the ears; calculated roughly with 3 ms per meter. This sums up to the developer goal that a good "ultra-low latency" audio transmission must get to below 10 ms (even better, below 8 ms). This is in accordance with the specs of one of the first wireless DJ headphones on the market [6]. Manufacturer AlphaTheta (ex Pioneer DJ) promises 9 ms.

The Theory...

Some months ago, the new Bluetooth LE Audio technology raised my interest, and I ordered some early adopter boards from the far east. But it soon turned out that even the low latency of 20 to 30 ms is not good enough, and furthermore, the audio must undergo compression, which reduces quality. But the fire was lit, and soon I discovered a lot of interesting "maker" projects on the internet for wireless audio transmission. Over the next few weeks, I had fun and learned a lot when experimenting with the feature-rich and well-documented Arduino Audio Tools from Phil Schatzmann [7]. He offers many examples for wireless audio via Wi-Fi, ESP-NOW, and others on his GitHub account. Especially ESP-NOW [8] raised my interest. This protocol is made by ESP32 manufacturer Espressif, and it uses the Wi-Fi PHY-layer which is, so to speak, built into the ESP8266 and ESP32 controllers, but without having the huge Wi-Fi overhead. On the internet, you can find many good Arduino examples using ESP-NOW. Most of them are used to transmit sensor data, but nevertheless, you can learn a lot with these.

If you know how to send packets, and you get familiar with the Espressif I2S API for sampling and playing audio (see the first article [1]), the step to stream audio continuously is not so big. As I already wrote in the first installment of this article, in the first instance, you don't have to think about the timing yourself. On the transmitter side, the following naïve approach works:

```
void loop() {
    I2S_ReadBytesInBuffer(byteBuffer,
                          packetLengthInBytes);
    Radio_WriteBytesFromBuffer(byteBuffer,
                          packetLengthInBytes);
    // here you can react on button presses
}
```

The reason is that `I2S_ReadBytesInBuffer()` (which is a wrapper of the Espressif `i2s_read()` sampling audio function) is blocking until the next packet of audio bytes is available from the ADC. Then you simply can send out the data with ESP-NOW or another protocol of your choice (here encapsulated by the `Radio_Write...` function).

On the receiver side, you have to output the audio when a packet is available:

```
if( Radio_DataReceived() )
{
  Radio_ReadSamplesInBuffer(mBuffer,
                packetLengthInSamples);
  I2S_WriteSamplesFromBuffer(mBuffer,
                packetLengthInSamples);
  // here you can do something
  // without wasting too much time
}
```

In this way, sampling the audio with a certain sampling frequency sets the timing for sending the packets. When all packets arrive in

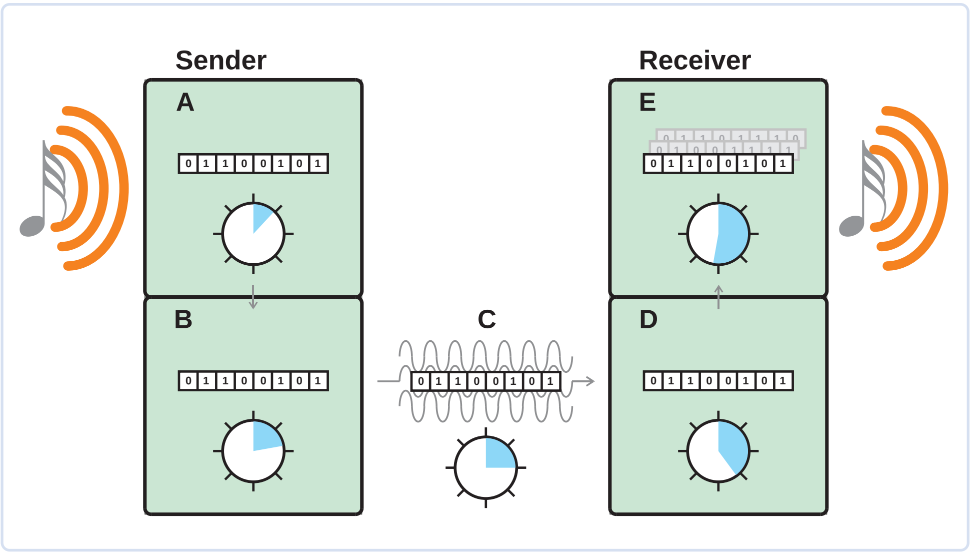


Figure 3: How latency adds up, in a simplified diagram. A) music is sampled and the digital data are collected to form a transmission packet, B) internal processing in the RF module, C) the transmission time itself is negligible, D) internal processing in the receiver module, E) music packets land in receiver buffers to balance timing differences. After all this, the first music sample is played.

time, and the transmitter and receiver use the same "master" system frequency for sampling and playing audio [1], you get a continuous, well-sounding audio signal on the receiver side.

...And Practice

ESP-NOW is a nice protocol. I was especially taken by the nominal datarates of 12 Mbit/s, 24 Mbit/s, or even higher. This means that a practical data rate of more than 1 Mbit/s is easily possible, and that means a transmission of uncompressed stereo music at 32 kHz and 16 bits (that is nearly CD quality). ESP-NOW makes it possible to send out packets of up to 250 bytes, but, from my experience, a packet size of 64 or 128 bytes is optimal for low latency. In principle, a packet can be sent out and received by the counterpart in about 4 ms, which is significantly better than Wi-Fi. For that reason, you can use ESP-NOW for synchronizing two or more ESP32s, but that is something for another article.

However, when it comes to reliability, it gets tough. We have to send out the audio data packets and hope that each of these reaches the receiver, unaltered and in time to not let the I2S DMA buffers [1] on the receiver side run empty, because this will be clearly audible. With a packet size of 128 bytes and 4 bytes per sample (16-bit stereo) we will have 32 music samples per packet, and, at a 32 kHz sample rate, that comes down to 1,000 packets per second. If only one of 100,000 packets gets lost, you would hear a glitch every 100 seconds, which is far too much, of course.

Unfortunately, ESP-NOW uses the same PHY-layer as Wi-Fi; as physical channels to select, you have the Wi-Fi channels 1 to 14. You can imagine that, in an apartment complex with several apartments, you have traffic on all of these channels, and that interferes with the ESP-NOW communication. A smartphone app such as WiFi Analyzer [9] can help to detect relatively empty channels, but the situation can change rapidly.

I did my first experiments with breadboards, but you will catch a lot of interference with the breadboard wires. I often heard ringing sounds mixed with the music, buffer underrun sounds, or simply nothing, and when I touched the wires, the music returned. Shielding with a metal housing and using an external Wi-Fi antenna connected to the XIAO ESP32 board (which I used for the ESP-NOW experiments) yielded better results in the end.

But still, a problem is the uncertainty of the transmission. Some packets arrive a bit later than they should, which you can alleviate with a bigger DMA buffer on the receiving side, but that immediately affects the latency (**Figure 3**). I used an oscilloscope and a simple waveform to measure the overall latency (**Figure 4**). In the end, I achieved a reliable connection in a Wi-Fi-like range of several meters, with a latency of about 20 ms. With a reduced buffer, 12 ms is possible, but then you have a very sensitive connection. I went late at night to a park in my neighborhood, but still, only the "20 ms modes" led to a reliable connection, and the maximum distance was about 8 to 10 m, which is not really adequate.

Nordic nRF24

I was about to give up the idea when my engineering friend Marco recommended looking for cheap analog radio modules at AliExpress instead. When googling for that, I discovered the Nordic nRF24 transceiver chips [10] accidentally. The manufacturer does not recommend using them in new designs, but they are still available in abundance, and there are many cheap breakout boards, with a common connector pinout. As already mentioned in the last installment, there are breakout boards that integrate, besides the chip, an antenna amplifier and an SMA connector for an external antenna.

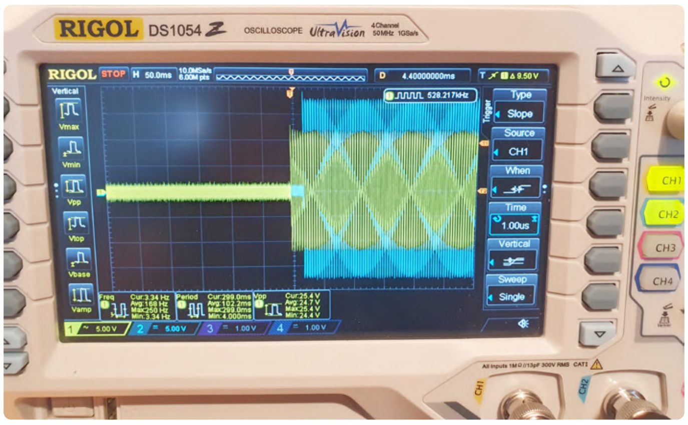


Figure 4: Measuring latency with a pulsed sine wave.

As the datasheet [11] says, nRF24 uses the manufacturer's proprietary wireless protocol in the 2.4 GHz band. The nRF24 communicates with a microcontroller via an SPI interface (MISO, MOSI, SCK, CS); additionally, you need to control a CE (chip enable) wire; and there is an IRQ pin that goes to low when data packets arrive (among other events). Of course, it would be possible to access the Nordic chip's registers with Arduino SPI transfer commands, but for that case, I opted for an external library; a good one comes from GitHub user TMRh20, called *nRF24* [12]. By the way, this user also made a library for audio streaming via nRF24 chips. However, according to him, that library is dedicated to AVR controllers from Microchip.

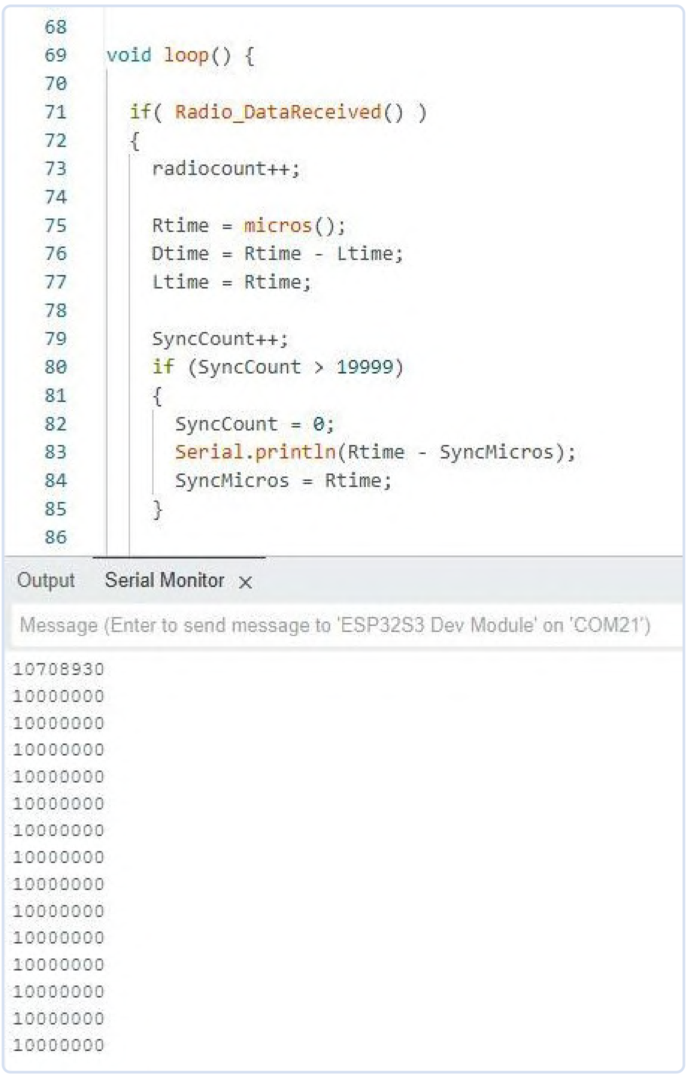
A disadvantage of the nRF24 protocol is the maximal packet size of 32 bytes, and another one is the highest nominal data rate of 2 Mbit/s. Like with ESP-NOW, the practical data rate I could achieve was far below that, first 500 kbit/s, later around 1 Mbit/s at the max. In principle, the latter enables the transmission of an uncompressed 16-bit/32 kHz stereo audio signal. However, the project presented in this edition, with a fairly good and stable connection, works with 16-bit/32 kHz audio, but in mono. That means, for a stereo connection, you have to build two transmitters and two receivers. For my purpose, I have to build multiple mono receivers anyhow; I can live with two independent transmission lines. The nRF24 can be configured to separate 125 radio channels with only 1 MHz bandwidth. From my experiments, channels must have a gap of about 5 MHz so as not to interfere with each other.

Timing and Latency

The maximum packet size of 32 bytes means you have to send out 2,000 packets per second. That is a lot. Nevertheless, it is possible, and in principle, you can use the same simple software approach as described above. In the code I wrote, I measured the time in microseconds between the packets arriving. The packets always arrive after 500 µs on average, and if you sum up over 20,000 packets, the precision is remarkable (**Figure 5**). However, between subsequent packets, there are timing differences, and these can change. We will come to that later.

The good thing about the Nordic chips and the technology is the very low latency. I have seen diagrams on the internet showing that the nRF24 can send a packet from transmitter to receiver in less than 1 ms. From my experience, packet transmission reliability is also better than with Wi-Fi and ESP-NOW, at least if you use a good nRF24 breakout board with antenna and antenna amplifier, a mode with maximum transmission power, and a shielded housing. Putting the antenna directly on the connector leads to far better results than using a small antenna cable. For that reason, we made the Audio Transceiver Board with the SMA antenna connector reaching over the outline of the PCB [1].

In the end, I could achieve reliable transmission throughout my whole apartment with walls in between; and in the field, there at least 20 m is possible. Due to that good situation and the stable transmission timing, I could reduce the number of I2S DMA buffers to 2 and the size to 32 samples. With that, a latency of around 5 ms is possible!



```

void loop() {

  if( Radio_DataReceived() )
  {
    radiocount++;

    Rtime = micros();
    Dtime = Rtime - Ltime;
    Ltime = Rtime;

    SyncCount++;
    if (SyncCount > 19999)
    {
      SyncCount = 0;
      Serial.println(Rtime - SyncMicros);
      SyncMicros = Rtime;
    }

```

Output
Serial Monitor
Message (Enter to send message to 'ESP32S3 Dev Module' on 'COM21')

```
10708930
10000000
10000000
10000000
10000000
10000000
10000000
10000000
10000000
10000000
10000000
10000000
10000000
10000000
10000000
```

Figure 5: Measuring the time until 20,000 radio packets are received. The sender and receiver system clocks seem to be perfectly in sync. The first figure deviating from 10,000,000 µs is due to the start of the program.

Some Problems and Solutions

Before we come to the (demo) project for this article, a wireless mono audio transmission line with ultra-low latency, I have to say that the project is far from being perfect: it is a work in progress. I have already identified potential solutions to several of the following concerns, which are occasionally crude but functional. For some of the issues, I see some potential that it can be done better in the near future, and I hope that our community — you — can also contribute to that. There may inevitably be some problems we can't fully solve, so we must learn to accept them. This project will never be a high-end solution for the guy who likes to listen to classical music in his armchair.

Lost Packets

When the distance between transmitter and receiver gets too high, the receiver gets sensitive to interference and attenuation, and a very successful blocking object in the 2.4-GHz frequency range is the human body. In fact, I still have to test the transmission line with many people in between. To minimize the audible effects of interference causing lost packets, I have built in a simple mechanism in the software. If a packet comes later than 900 µs after the previous one, it is assumed that a packet got lost, and then, the I2S controller inside the ESP32 is restarted, followed by a delay of about 10 ms. Of course, that is audible as a short crack, but a very quiet one, so there is no disturbing sound. If two loudspeakers and a stereo transmission are involved (with two independent transmitters and receivers), or even more loudspeakers, short interferences are even less audible.

Additionally, I already did some experiments with a dual-channel approach. It is possible to connect two nRF24s to the ESP32, on the same SPI lines, but with different CE/CS pins. (You can use the Audio Transceiver Board for a prototype, when a 0.9-inch wide ESP32-S3 DevKit is plugged in, see **Related Products**.) In software, you can be fast enough to poll both nRF24s, configured to receive data on two different radio channels. A simple timeout mechanism decides if the packet of the first nRF24 arrived in time; if not, the packet received on the second channel can be used. The results I got are very promising. More on this will most likely follow in the next installment.

Singing

A very annoying issue that haunted me for weeks is a kind of ringing or "singing" sound coming out of the receiver. You still hear music, but there is a kind of overlaid high frequency sound, which often seems to go with the beat. This error can be triggered if you put the 16 samples arriving with the transmission packet in a delay buffer, but you always give just fresh 15 samples to the I2S DAC, and 1 sample is taken from a packet which arrived earlier.

In fact, in early stages of the project, I heard this "singing" sound often. And it came mostly several minutes after starting the receiver. As this issue occurred quite regularly with one and the same receiver — for example, always after 4 minutes plus/minus 5% — I first thought it may be a synchronization issue, due to differences in the system clocks of the ESP32s used. My solution was a kind of soft restart of the I2S controller, after a certain period (for example every 2 minutes). For that, I have built in a kind of detection of a very quiet music sample, to process the restart then. In a lot of cases you cannot hear the restart then at all.

However, with the latest version of the prototype and the software, I could not reproduce this "singing" issue again, so my first assumption wasn't correct. But as mentioned, with the new PCBs (and the antenna connector at the right place) I could not hear that annoying sound anymore. ESP32 transmitter and receiver going out of sync may nevertheless still be a problem. So I kept the "dirty reset" in the software, but I have set the rhythm to acceptable 1,000 s.

The Stereo Sync

As I need stereo, I have always tested two transmission lines, meaning two transmitters and two receivers in parallel. In my home office, which also my lab, I have some good studio monitors to check the sound quality. Occasionally I had the feeling that the stereo effect is slightly shifting, sometimes the sound was more in the center, sometimes the sound seemed to be a bit "wider." By the way: You can hear that quite well if you put pink noise on both audio channels.

After some investigation, I got an idea what could be the reason for that. On average, packets arrive after 500 µs; but if you measure the time between subsequent packets, you get a curve as in **Figure 6**. If you use the simple software algorithm for the receiver described above, timing differences can even pile up. Furthermore, the difference between late and early packets can change suddenly. If the situation for the left and right channel receivers is different, this will affect the stereo image.

In a second software version for the receiver, I decoupled two things in time: polling the radio module and outputting the samples. For that, I used an extra, short ring buffer. This software approach is more professional, and offers options like the dual-channel operation I mentioned above. With the ring buffer, you can also put in an artificial delay, which may have some use in the future.

It is still not a perfect solution, though; From time to time, I still experience subtle changes in the stereo image. This has to be investigated further, and maybe we need some sophisticated receiver synchroni-

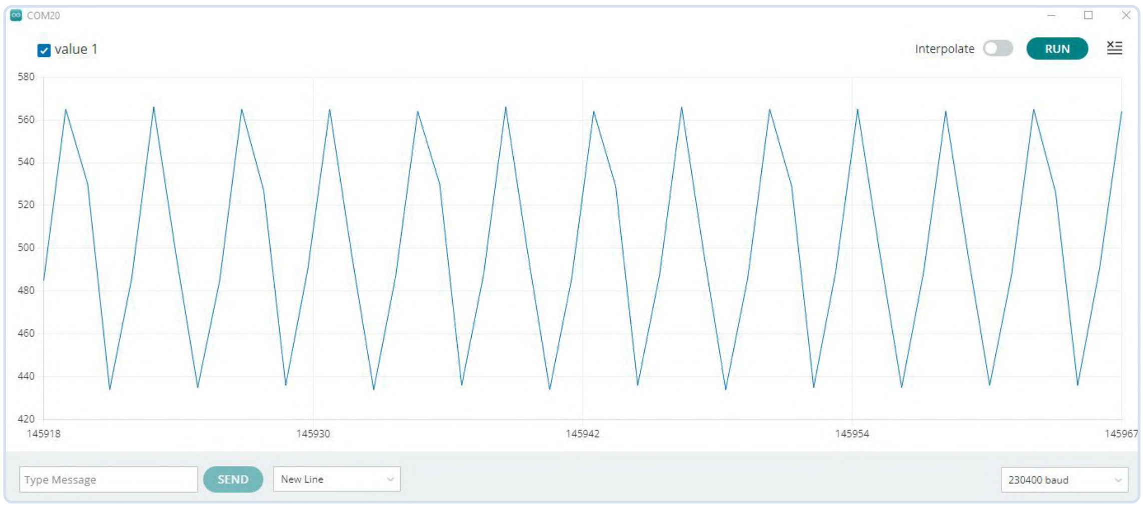


Figure 6: On average, a packet arrives after 500 µs in total, but there are timing differences between subsequent packets.

zation using time stamps or the like. But for now and for my use case, I can live with the results.

Busy Radio Channels

As previously mentioned, the ubiquitous Wi-Fi routers have only a minor effect on the transmission, but there is still some, and of course there are also a lot of other wireless devices in the 2.4 GHz band. So there had to be an option to select another nRF24 radio channel in the field. That was an easy one to solve. I opted for a semi-automatic solution; if you change the channel at a transmitter manually, and the receiver will follow. The command to change the channel is transmitted instead of one of the music packets. After that, we have to restart the whole system anyhow. As I am using two independent transmission lines, I reserved some of the radio channels for the left audio channel and some for the right one. At the moment, the audio channel for the device (meaning the default radio channel) is hardcoded in software, but you could also use a jumper or DIP switch for that.

Data Rate

As mentioned earlier, the radio modules used and the nRF24 library enable practical data rates of 1 Mbit/s, which corresponds to a 32 kHz/16 bit stereo transmission on only one radio channel (and with one transmitter). However, this results in 4000 messages per second instead of 2000! At the time of writing this article, I was working on such a connection without sound artifacts. Work in progress!

Demo Hardware

In **Figure 7**, **Figure 8**, and **Figure 9**, you can see my transmitter and receiver prototypes. On both sides of the transmission line, the Audio Transceiver Board, plus the standard modules and the metal housing, are used, which were described extensively in the last installment of this article series.

For the transmitters, I drilled one hole at the front for the antenna SMA connector, and four holes at the back. A 12-mm one is used as a window to show at least one digit of a 7-segment display (I used one from the Grove system). That number is used to represent the radio channel and the audio channel (see Figure 7). The numbers 1, 3, 5, 7, and 9 stand for five radio channel options for the left audio channel. 0, 2, 4, 6, 8 stand for five different radio channels used to transmit the right audio channel. Another 12-mm hole is used for a USB-C connector; however, this is just a 2-wire type for the power supply. Smaller holes are used for a cinch connector and a button to change the channel.

On the receiver side, I drilled an additional 5-mm hole in the front for an LED. It lights up when an I2S reset is done because of a lost packet, and also at a “dirty reset” — or when a command for switching radio channels is received. As you can see in Figure 9, the prototypes are not fully finished on the backside. At the moment, there is just an audio and a USB cable running through two holes.

Software Libraries

Of course, there is firmware for the transmitter and for the receiver. You can download both at [13]. However, they have many things in common; for example, both use the same *Stream_I2S.h* and *Radio_Nordic.h* libraries, which handle sampling and playing audio via the I2S

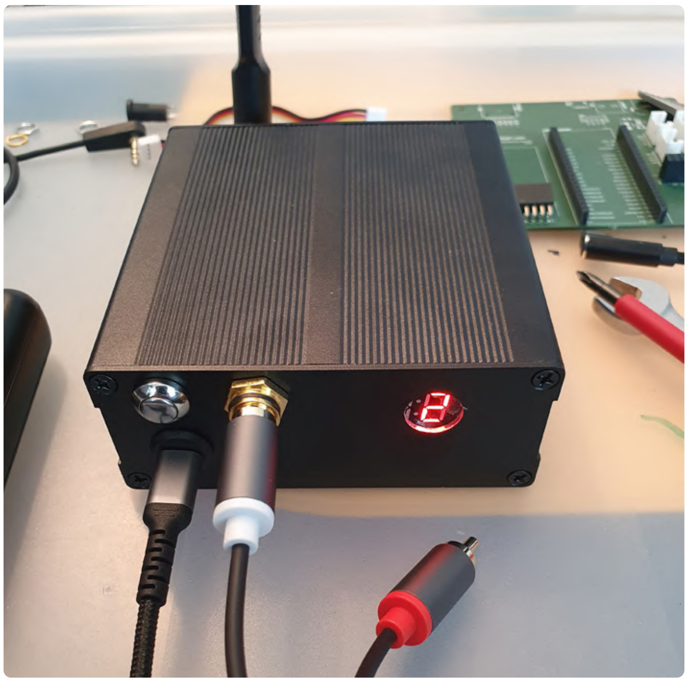


Figure 7: Prototype of the transmitter, with a cinch input, a button to switch the radio channel, and a seven-segment display for showing the radio channel index (0 to 9).

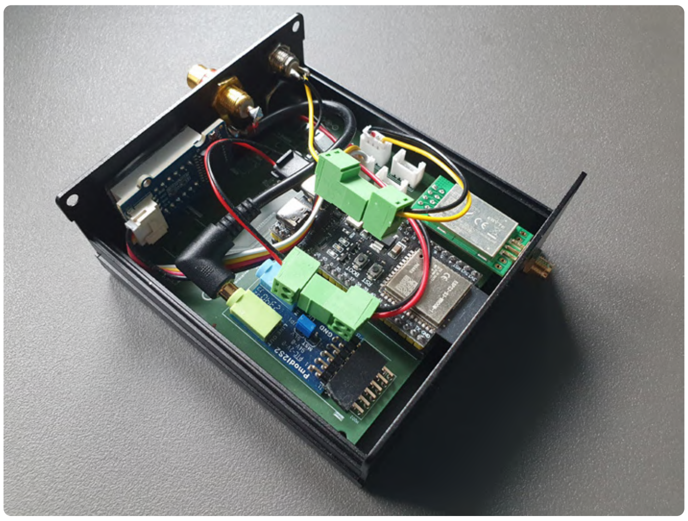


Figure 8: The antenna connector of the nRF24 module goes through the metal housing.

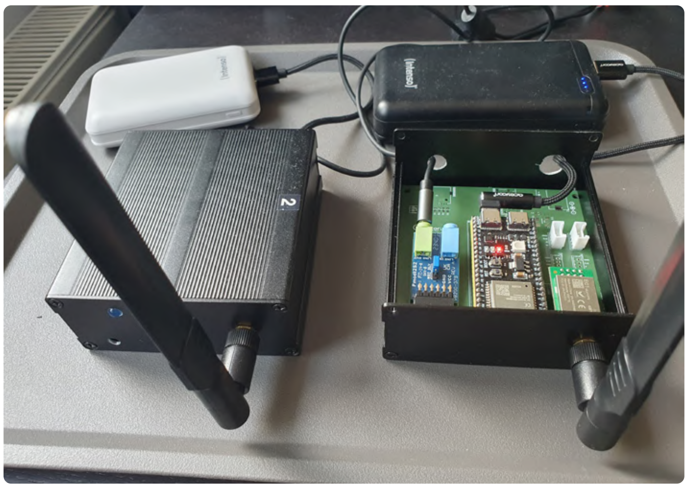


Figure 9: Prototypes of two receivers.

interface, and transmitting and receiving data packets via the nRF24 modules. As I already said, the libraries stand on the shoulders of the Espressif I2S API and the *nRF24* library by TMRh20. However, I tried to further simplify and unify audio sample packet-handling, which can also be used for other data packets. Functions of the form

```
xxx_WriteSamplesFromBuffer(int16_t* wBuffer,
            uint16_t NumberOfSamplesToWrite)
```

output (send out) audio packets; the first parameter is a pointer to a buffer, and the second parameter is the number of samples to output.

With the functions of the following form, you can input (receive) audio samples:

```
xxx_ReadSamplesInBuffer(int16_t* rBuffer,
          uint16_t NumberOfSamplesToRead)
```

There are also corresponding functions to write, read, send and receive bytes instead of samples. Presently, the ratio is fixed to 1 sample = 2 bytes, so we are always dealing with 16-bit audio resolution. One thing I have to mention is that I am expanding the mono signal on the receiver side to a two-channel output (this is done in *Stream_I2S.h*), because this has proved to work more stable. Of course, this is not real stereo; the same mono signal is output on both of the receiver's stereo channels.

In *Boardpin.h* you can find pin definitions for the board and the modules connected to it. I also developed generic libraries for button presses and the visualization of values. Read more on these and the principle behind it in the **User Interface Abstraction** textbox.

User Interface Abstraction: Input and Output Libraries

The *Input....h* and *Output.h* libraries abstract input and output elements, which means the user interface. Imagine you have written good firmware, but later you would like to use a seven-segment display instead of two LEDs; or instead of pressing a button for a short duration or a long one to trigger two different actions, you would like to have two buttons. This can be handled by changing the *Input* and *Output* libs, whereas the main program can stay the same.

In case of the *Output* library, the main program just says what should be visualized, and not how this should be done. In the case of our audio sender, it would be, for example, the number of the current radio channel, and if an error occurred:

```
Output_SetValue(OCHAN_ERROR, 0);
Output_ShowValue(OCHAN_RADIOCHANNEL,
RadioChannelIndex);
```

In that case, the main program tells to the *Output* lib, what values should be shown, via different "output channels" (`OCHAN`). Here we need two of them. But there is another difference between the two code lines. The `_SetValue()` function just sets the value to show, whereas the `_ShowValue()` function sets a value and refreshes the user interface.

For reading input values based on button presses, such as the turning of a rotary encoder, the main program calls the `Input_Loop()` function continuously, which will update these values. With the `Input_GetChannelOfChangedValue()` function, the main program can request which values were changed, to trigger the appropriate action. This decouples the main program from the actual input elements of your system.

These libraries are also a work in progress. Maybe we can dive deeper in one of the following installations.

The Main Program

With all that, the transmitter's main program is easy to understand. It starts with some `#define`s for the audio channel and the packet length, which are also used in the libraries. After including the libraries, we define the buffers needed. In the respective `setup()` function, we initialize the radio module and the I2S controller.

The first lines in the `loop()` function speak for themselves:

```
I2S_ReadBytesInBuffer(byteBuffer, PACKET_LENGTH_BYTES);
Radio_WriteBytesFromBuffer(byteBuffer,
                PACKET_LENGTH_BYTES);
```

After that, we have enough time to check for a button press. A short one triggers finding the next radio channel, send out the corresponding message to the receiver, and then change the transmitter's radio channel.

The main program for the receiver is a bit more sophisticated. After including the libraries, a buffer is defined, but in that case a buffer of 16-bit integers:

```
int16_t mBuffer[PACKET_LENGTH_SAMPLES];
```

The reason that we just don't use a simple byte buffer is that we want to get easy access to the received music samples.

In the `setup()` function, two variables are set, which define the timeout in microseconds and the time for a dirty reset of the I2S controller:

```
TimeOutReceiving = 900;
dirtyResetSeconds = 1000;
```

The magic starts in the `loop()` function. We regularly poll if some data packet is received. If so, the time from the last packet arriving to the current one is measured. Then we decide if we have a timeout, which means that the I2S controller is restarted. This would of course again produce a lost packet, so there is a simple mechanism, that only a timeout after a packet coming in triggers the restart.

After this, we read the data of the packet in the `mBuffer`. Furthermore, we copy the first 8 bytes into a buffer of bytes:

```
Radio_ReadSamplesInBuffer(mBuffer,
            PACKET_LENGTH_SAMPLES);
memcpy(&bBuffer, &mBuffer, 8);
```

This helps to investigate if a command to change the radio channel was received. To discriminate such a command from music samples, the first three bytes of the packet must be 255. If a command arrives, and the audio channel on the transmitter is the same as the one on the receiver, the radio channel is changed.

Thereafter, we handle the regular ("dirty") reset. First, we look if the packet arrived just contains quiet music. When this is true and it is time for the reset the function `I2S_Reset()` to be called, which causes a kind of soft start for the I2S controller (without reconfiguration with new parameters).

At the very end of the `loop()` function, we output the music:

```
I2S_WriteSamplesFromBuffer(oBuffer,
            PACKET_LENGTH_SAMPLES);
```

In the second, timer-based version of the software, this is done every 500 µs. For the timer microseconds, I put in a number of about 480 instead of the full 500, as 500 does not work properly. This may be due to the fact that the Arduino `micros()` function is not precise. With low-level ESP32 timer programming, there could be some further improvements.

To Be Continued...

Many things are still to be told and investigated. You may be interested, for example, in the power consumption of the transmitter and the receiver, respectively, and maybe also in other sampling rates. Meanwhile, I also built a device to measure music latency, also based on the Audio Transceiver Board. More on this and, of course, further developments will follow in one of the next editions. As always, please don't hesitate to send me hints, ideas, and your experiences and improvements. Stay tuned to music!

250384-B-01

Questions or Comments?

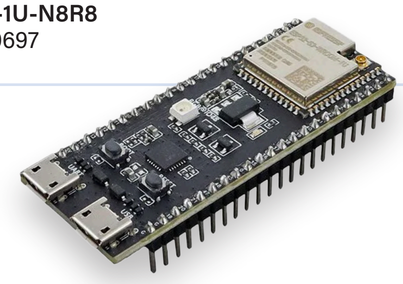
If you have questions about this article, feel free to email the Elektor editorial team at editor@elektor.com.

About the Author

Jens Nickel studied physics and was further educated as an editor for professional tech/science magazines. In 2005, he started working for Elektor, and for the past several years, he has served as the Editor in Chief of the magazine. Jens has liked programming since the early days of the C64. No wonder that he is also fascinated by the flexible and powerful options of good microcontroller platforms. Over the last few years, the ESP32 has become his favorite controller for personal projects. Jens's hobbies include music videos, digital audio, and studio control projects.

Related Products

> ESP32-S3-DevKitC-1U-N8R8
www.elektor.com/20697



WEB LINKS

[1] Saad Imtiaz and Jens Nickel, "ESP32 Audio Transceiver Board (Part 1)," Elektor Circuit Special 2025: https://www.elektormagazine.com/250384-01
[2] AudioVideoStations - remote controlled wireless Loudspeakers, Elektor Labs: https://www.elektormagazine.com/labs/audiovideostations-remote-controlled-wireless-av-devices
[3] Analog Transmitter, Inda Audio: https://www.inda-audio.com/product/RM-UHF-Wireless-Transmitter-with-Mic-3-5-Channel-300-Meters.html
[4] Analog Receiver, Inda Audio: https://www.inda-audio.com/product/RF-UHF-Rechargeable-Wireless-Tour-Guide-System.html
[5] Bluetooth LE Audio: https://www.bluetooth.com/learn-about-bluetooth/feature-enhancements/le-audio/
[6] AlphaTheta Wireless Headphone with 2.4 GHz transmission: https://alphatheta.com/en/product/headphones/hdj-f10/black/
[7] Arduino Audio Tools, GitHub: https://github.com/pschatzmann/arduino-audio-tools
[8] ESP-NOW, Espressif API: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/network/esp_now.html
[9] Wifi Analyzer: https://www.wifianalyzer.info/
[10] Nordic nRF24, Nordic: https://www.nordicsemi.com/Products/nRF24-series
[11] nRF24L01+ Datasheet, Nordic: https://docs.nordicsemi.com/bundle/nRF24L01P_PS_v1.0/resource/nRF24L01P_PS_v1.0.pdf
[12] nRF24 Library, GitHub: https://nrf24.github.io/RF24/
[13] Software Download, Elektor Labs: https://www.elektormagazine.com/labs/esp32-audio-and-iot-board

Inductive AM Transmitter

Uses Pico's PIO in an Arduino Sketch

By Clemens Valens (Elektor)

Ten years ago, Elektor published a simple Arduino-based AM radio transmitter for streaming audio to vintage receivers. While functional, the original design lacked polish and robustness. In this updated version, we modernize the project using a Raspberry Pi Pico and explore how to integrate a PIO program into an Arduino sketch.

In 2015, Elektor published a simple circuit for an inductive AM transmitter that let you stream e.g. music to a vintage tube-based AM radio receiver [1]. Of course, it also worked with transistor radios as long as they have an AM radio band. The circuit used an Arduino UNO, an add-on shield with a display and a pushbutton, and a little circuit assembled on a piece of prototyping board. Even though this system worked fine, it didn't have the looks required for something to be placed in sight, in a living room or study or so. Also, it was not very robust. In this article we revisit the ten-year-old project and turn it into something more practical. While doing so, we learn how to integrate a PIO program in an Arduino sketch.

For those who are not familiar with PIO: it stands for Programmable Input/Output. The RP2040 microcontroller of the Raspberry Pi Pico has two PIO modules which can control most I/O pins at the Pico very fast and independant from the processor core; these blocks can be programmed with an assembly-like programming language [2].

Amplitude Modulation in a Nutshell

Let's start by explaining the working principle of the AM transmitter. AM works by multiplying a low-frequency input signal by a high-frequency carrier signal. The input signal is the audio signal; in the case of AM radio the carrier signal is a high-frequency sinewave in the range from 530 kHz up to 1.6 MHz. The result of the modulation, shown in **Figure 1**, is that the amplitude of the input signal is superimposed onto the carrier signal. To recover the

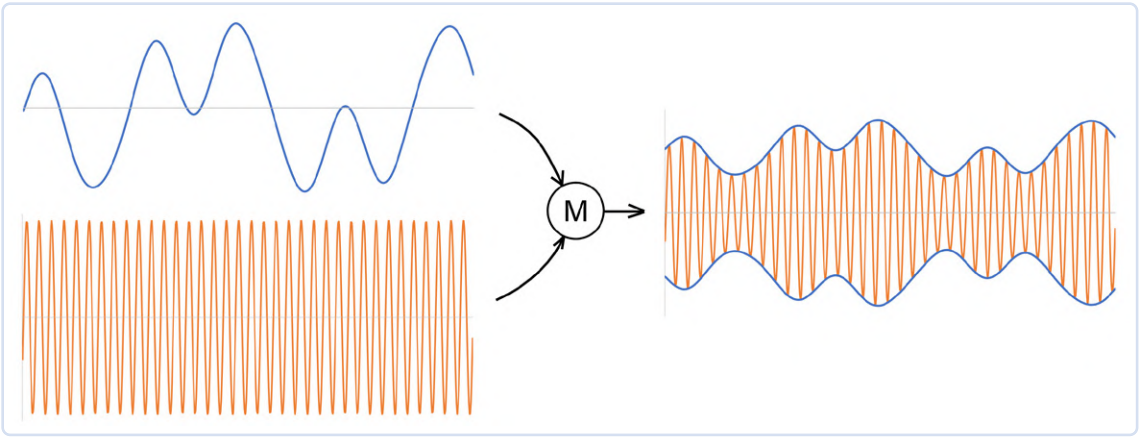


Figure 1: Amplitude modulation.

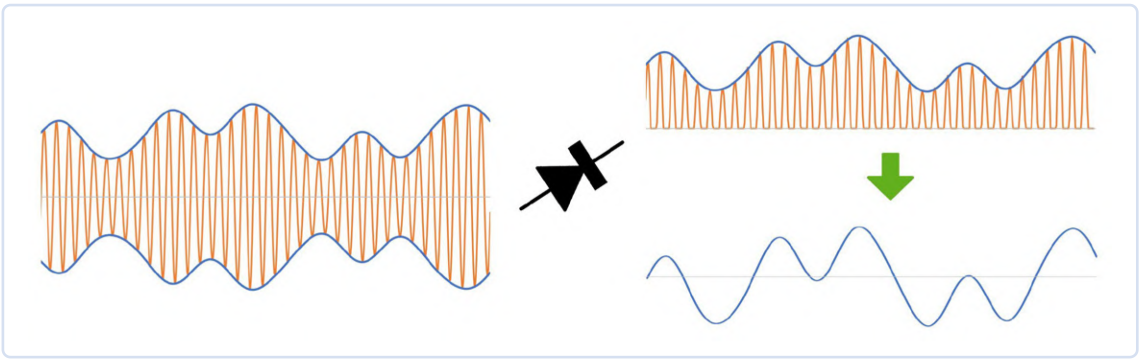


Figure 2: Amplitude demodulation a.k.a. detection.

input signal, it is enough to rectify the modulated carrier and filter out the frequencies higher than those present in the audio signal. This is called detection and is done by an AM receiver (**Figure 2**).

Poor-Man's AM

Generating a stable and programmable high-frequency sinewave is a bit complicated, but it may be approximated by a square wave. Doing amplitude modulation with a square wave instead of a sinewave will introduce unwanted harmonics, but they can be filtered out in a later stage (ideally before transmitting them). Generating high-frequency square waves is so much easier than generating sine waves that it justifies the effort of removing the harmonics afterwards. Unwanted harmonics in radio transmissions can cause interference with other frequency bands, potentially disrupting nearby communication systems and violating regulatory limits. In our case, the transmitter has a range measured in centimetres and so we don't really care much about the unwanted harmonics (just don't tell anyone about it), and the filtering can be kept to a minimum.

A New Microcontroller

The AM transmitter uses a microcontroller to generate the carrier frequency in the AM band, from 500 kHz up to 1.6 MHz in 32 steps of about 35 kHz. In the original design, the MCU was an ATmega328 AVR. The timers inside an AVR clocked at 16 MHz can generate square wave signals with frequencies up to 8 MHz, but they lack the granularity required to produce the 32 frequencies needed for our AM transmissions. Therefore, the AM carrier signal was generated by toggling a pin in an endless loop calibrated with NOP instructions for each frequency. The Arduino program had 32 of these loops, one for each frequency.

Even though the Atmega328 still is a great microcontroller and the Arduino UNO a practical platform for using it, today, ten years later, there are other solutions to achieve the same results in a somewhat more elegant way. Therefore, we replaced the Arduino UNO by a Raspberry Pi Pico as it is much smaller and faster, and also cheaper. Furthermore, we replaced the LCD by one of those small OLED displays that are so popular these days. The resulting schematic is shown in **Figure 3**.

The Circuit

The audio signal enters the circuit at K2. This is a stereo input, but AM is mono and so R1 and R4 add the two channels together. C1 superimposes the mono audio signal onto a 2.5 V DC level created by R5 and R6 while C2 provides some lowpass filtering.

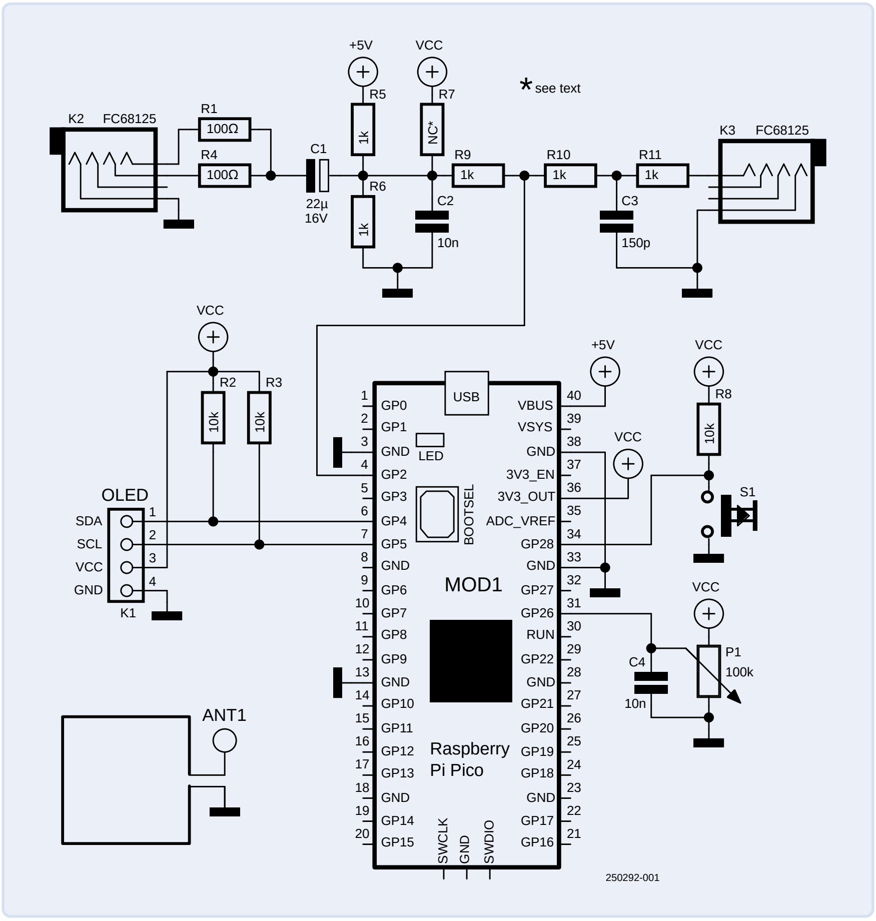


Figure 3: The inductive AM transmitter based on a Raspberry Pi Pico.

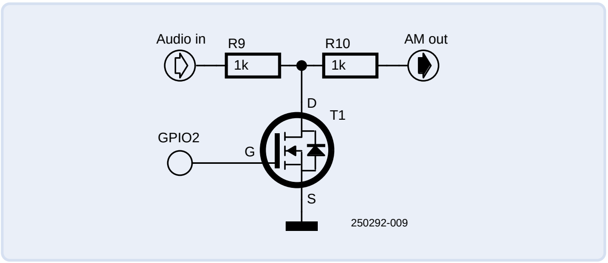


Figure 4: Poor-man's amplitude modulation.

Next comes the modulator. It isn't very visible, though. To see it, imagine node R9/R10 connected to the drain of a N-type MOSFET that has its source connected to ground and its gate connected to GPIO2 (**Figure 4**). The gate is driven by the carrier signal. When the gate is driven high, the MOSFET conducts and node R9/R10 is pulled to ground. When the gate of the MOSFET is driven low, the transistor blocks and node R9/R10 is pulled up to the current level of the audio signal. Consequently, the amplitude of the signal on node R9/R10 is superimposed on the high-frequency carrier. This simple trick works as long as the audio signal remains inside the boundaries imposed by the power supply. In other words, with the 5 V supply, the audio signal should not exceed ±2.5 V.

Note that the Pico does not support 5 V levels on its pins. In our case, R9 provides enough protection to allow connecting R5 to 5 V. However, if you feel this is unsafe, R7 allows you to connect the input voltage divider to 3.3 V. Keep in mind that doing so does limit the maximum input signal level to ±1.65 V.

Loop Antenna

The signal on node R9/R10 is passed through a simple 1 MHz lowpass RC filter (R10/C3) before being injected into the loop antenna. Note the word 'loop' as the antenna is a length of wire connected between the output and ground. Such an antenna can be connected to K3 with a 3.5 mm mono jack.

The rest of the circuit is the microcontroller that generates the carrier signal and that provides the user interface. The OLED display on K1 is connected to the I²C bus of the Raspberry Pi Pico. Potentiometer P1 is the tune control that allows selecting one of the 32 frequencies. S1 is used to toggle transmission on and off.

PIO Programming

The Raspberry Pi Pico's RP2040 microcontroller's Programmable Input/Output (PIO) peripheral is used for generating the programmable carrier frequency. The PIO is perfectly suited for generating square wave signals with precise frequencies in the AM band, all that is needed for this is a 4-line PIO loop (see below). The PIO itself is controlled from the Arduino sketch. This brings us to the following interesting topic: How to use the PIO in an Arduino sketch?

Using the PIO in an Arduino Sketch

Programming the Raspberry Pi Pico is very comfortable when it is done with the Arduino IDE and Earle Philhower's Raspberry Pi Pico Arduino Core [3]. However, using the PIO in this environment is a bit more complicated as it involves a special tool needed to compile the PIO program that is not accessible from the IDE. So how do you do this? Easy:

1. Write the PIO program according to the specifications and instructions provided by Raspberry Pi.
2. Compile the PIO program into a header file using the *pioasm.exe* tool included in Earle Philhower's RP2040 Boards Package.
3. Include this header file in the Arduino sketch.
4. Use the PIO program as any other PIO program.

It is Step 2 that makes this easy. Why? Because for some reason, in the official Raspberry Pi Pico development environment, the *pioasm* program must first be built by the user before it can be used to compile a PIO program. Earle Philhower's package however includes a precompiled version of *pioasm*, which removes the extra compilation step. Note that we included Earle Philhower's *pioasm* in the download so you don't have to search for it.

PIO-Based Frequency Generator

The PIO program looks like this:

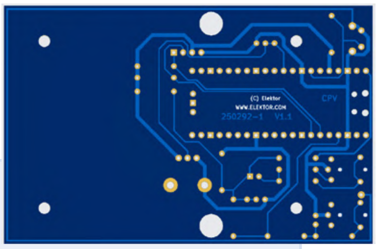
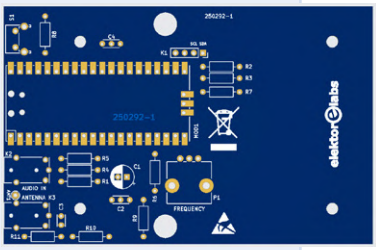
```
.program my_pio

   set pins, 0      ; Pin defaults to 0
loop:
   set pindirs, 1  ; Set pin to output
   nop             ; Wait 1 cycle to compensate for jmp
   set pindirs, 0  ; Set pin to input
   jmp loop
```

Component List

Resistors
R1, R4 = 100 Ω
R2, R3, R8 = 10 kΩ
R5, R6, R9, R10, R11 = 1 kΩ
R7 = NC (see text)
P1 = potentiometer 100 kΩ, linear

Capacitors
C1 = 22 µF 16 V, 2.5 mm pitch
C2, C4 = 10 nF, 2.5 mm or 5 mm pitch
C3 = 150 pF, 2.5 mm pitch

Miscellaneous
K1 = 4×1 pin socket, 0.1" pitch
K2, K3 = 3.5 mm socket FC68125
MOD1 = Raspberry Pi Pico
S1 = pushbutton, angle mount
0.96" monochrome I²C OLED display

All it does is toggle the direction of pin GPIO2. The pin's number is specified at "installation" time of the PIO program during setup by `out_pin`:

```
#include <hardware/pio.h>
#include "my_pio.h"
PIO pio = pio0;
uint offset;
uint sm;
const uint8_t out_pin = 2;
float frequency_hz = 1000000; // Start-up frequency.

void setup(void)
{
...
   // Setup PIO stuff.
   offset = pio_add_program(pio,&my_pio_program);
   sm = pio_claim_unused_sm(pio,true);
  my_pio_program_init(pio,sm,offset,out_pin,frequency_hz);
...
}
```

The frequency of the output signal is adjusted by the PIO's clock divider in the function `update_frequency`. With the 4-cycle PIO program and the clock divider, about any frequency in the range from 763 Hz up to 50 MHz can be generated (supposing a 200 MHz system clock).

Note that the program doesn't toggle the pin's level, but its direction. The pin is set to low before entering the loop. When the pin is configured as an output, it will be low and node R9/R10 will be low. When it is configured as an input, it behaves as if it wasn't there (i.e., as an open drain). In this case, node R9/R10 is pulled up to the current level of the audio signal.

If, in the PIO program above, the two `pindirs` are replaced by `pins`, and adapting the sketch accordingly, the program can be turned into a simple square wave signal generator with a wide frequency range (an example is included in the download [4]).

The Arduino sketch takes care of the user interface. It reads the voltage set by potentiometer P1 and turns it into one of 32 frequencies. The display shows the selected frequency. As soon as S1 is pressed, the program enters *Transmission Mode*. This fixes the frequency of the carrier signal and starts outputting it on GPIO2. Press S1 again to return to idle mode.

Antenna On-Board

We designed a printed circuit board (PCB) for the AM Transmitter. It is single-sided and fits inside a cheap Hammond 1593N enclosure (available in black BK, grey GY, and translucent blue TB, see **Figure 5**). Single-sided-ness was achieved by using more than one ground pin of the Pico, therefore, make sure to connect them where needed.

A rectangular loop antenna of about 10 cm by 6 cm is included on the board. It is not the best antenna, but it does allow coupling of the AM signal into the antenna of a receiver placed close by. We found that placing the short, far end in parallel to the radio's ferrite antenna produced excellent results. To use the on-board antenna, connect pad ANT1 to the hole of K3 closest to it. Better or other antennas can be connected to K3 but don't forget to disconnect the on-board antenna.

Downloads for this project are available at [4].

250292-01

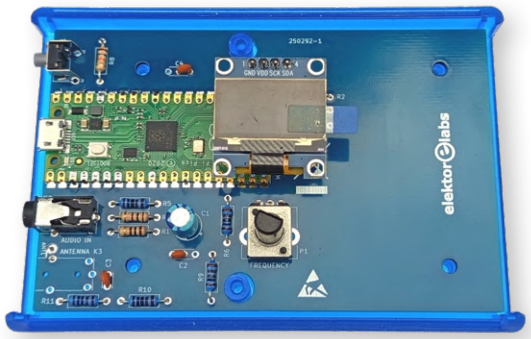


Figure 5: The assembled AM Transmitter board inside a suitable enclosure.

Questions or Comments?

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

Related Product

- **Clemens Valens, *From Rubbing Amber to Swiping Glass* (Elektor)**
  www.elektor.com/21204

WEB LINKS

[1] Burkhard Kainka, "Arduino-Powered AM Transmitter," Elektor 5-6/2015: https://www.elektormagazine.com/magazine/elektor-201505/27787
[2] What is PIO?, Raspberry Pi: https://www.raspberrypi.com/news/what-is-pio/
[3] Earle Philhower, Raspberry Pi Pico Arduino Core, GitHub: https://github.com/earlephilhower/arduino-pico
[4] Downloads for this project, Elektor Labs: https://www.elektormagazine.com/labs/am-transmitter

Navigating Wireless Protocols

A Technical Guide

By Saad Imtiaz (Elektor)

In embedded systems, choosing the right wireless protocol is often a fundamental design decision. The choice directly affects performance, power consumption, range, reliability, and software complexity. In this article, we look at the most common protocols — Wi-Fi, Bluetooth, ZigBee, LoRa, ESP-NOW, and some more — and examine how they work, their pros and cons, and where each one makes the most sense.

In many embedded, IoT, and home automation projects, devices need reliable communication to exchange data efficiently. Wired protocols such as Ethernet, emerging single-pair Ethernet, CAN, and traditional RS485 often remain the top choice because they offer excellent reliability, security against interference or spying, and simplify powering multiple devices from a centralized supply. However, wireless communication also brings significant advantages. It eliminates the complexity and cost of running cables, particularly in locations where wiring is difficult or impossible. Moreover, with independent power sources, wireless devices can be truly portable and mobile, opening new possibilities for project designs. In this article, we explore some of the most common wireless protocols popular in home automation, IoT applications, and maker projects, providing a helpful (but by no means exhaustive!) guide for selecting the right technology for your needs.

Wi-Fi (IEEE 802.11)

Wi-Fi is a high-bandwidth wireless communication standard commonly used in consumer electronics and embedded IoT devices. It operates primarily in the 2.4 GHz and 5 GHz bands and supports IP networking natively, making it ideal for devices that require direct internet access or integration with web services. Wi-Fi offers high data rates, typically ranging from 10 Mbit/s to over 100 Mbit/s on newer embedded chipsets like ESP32. For instance, the newer ESP32-C5 supports Wi-Fi 6 and dual-band operation (2.4 GHz and 5 GHz) [1], reaching tested speeds around 70 Mbit/s. However, theoretical Wi-Fi 6 speeds can reach up to 9 Gbit/s on advanced routers.

Emerging Wi-Fi standards such as Wi-Fi 7 further push this envelope, with theoretical speeds up to 46 Gbit/s, and Wi-Fi 8 is expected soon, targeting groundbreaking speeds around 100 Gbit/s. These extremely high speeds are not achievable in embedded systems due to hardware constraints like limited processing power, memory, antenna designs, power availability, and the absence of multi-stream MIMO configurations.

Embedded devices typically prioritize lower power consumption and cost efficiency over maximum throughput. Hardware commonly used in embedded systems, such as the ESP32 by Espressif, and Raspberry Pi, focuses on balance between performance and power efficiency, making them suitable for applications in IoT sensors, and remote monitoring devices. Typical indoor range is around 30 to 50 m but can be substantially affected by walls and interference, while outdoors it can reach 100 m in clear conditions.

Overall, while Wi-Fi's main advantages are high throughput, mature software ecosystems, and broad compatibility, its downsides include relatively high power consumption and potential network congestion. Wi-Fi works best in environments with stable power sources and is common in smart home hubs, IoT gateways, and cameras. Although Wi-Fi provides high data throughput and easy network integration, its power demands and interference issues in crowded networks can be limiting factors.

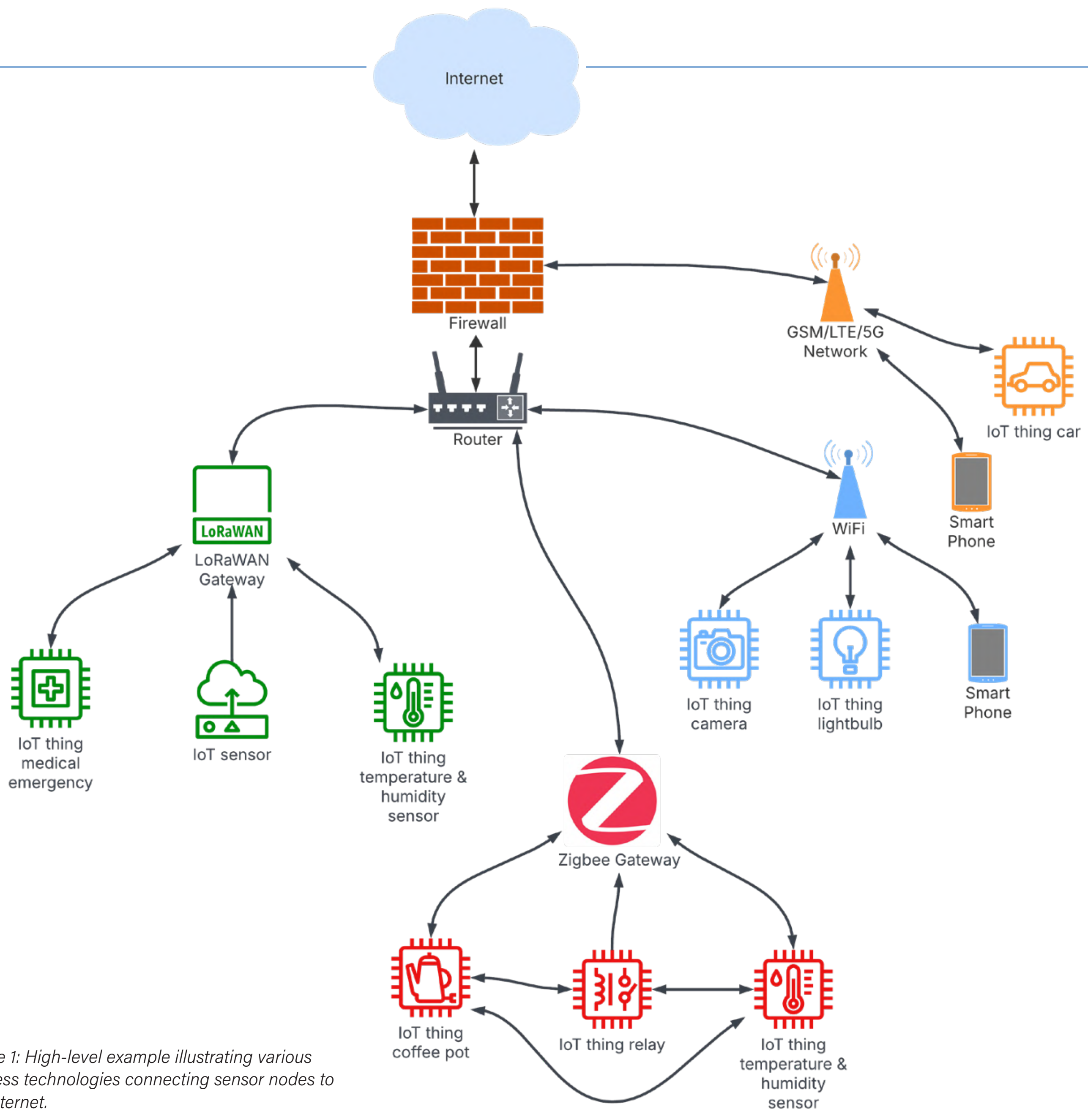


Figure 1: High-level example illustrating various wireless technologies connecting sensor nodes to the internet.

Bluetooth (BLE)

Bluetooth, especially Bluetooth Low Energy (BLE), is designed for short-range communication with low-power consumption. Operating in the 2.4 GHz band, BLE typically achieves data rates around 1 Mbit/s, which is sufficient for most wearable devices, sensors, or smartphone accessories.

Typical BLE hardware includes the Nordic nRF52 series [2], ESP32 modules, and DSD TECH HM-10 modules. BLE devices can operate for months or even years on small batteries, with indoor ranges usually around 10 meters and outdoor line-of-sight distances up to 30 meters. BLE's strengths include excellent power efficiency, quick connectivity, and strong mobile device support. Its main drawbacks are limited range, modest data throughput, and vulnerability to interference in crowded 2.4 GHz environments. BLE is most suitable for wearable health monitors, wireless sensors, and any application requiring low-power mobile connectivity.

LoRa, LoRaWAN, and Sigfox

Long Range (**LoRa**) is a modulation technology designed for extremely long-range, low-data-rate wireless communication, typically used in sub-GHz bands (e.g., 868 MHz or 915 MHz). LoRa can achieve ranges of several kilometers, often reaching up to 10 km in rural or open environments.

Typical hardware includes the Semtech SX127x series [3], modules from Microchip [4], STMicroelectronics [5], Seeed Studio [6], Ebyte [7], RAK Wireless [8] and more. Data rates are low, ranging from 300 bit/s to about 50 kbit/s, depending on configuration. Power consumption is exceptionally low, making LoRa devices capable of operating on batteries for many years.

LoRaWAN, a network layer built on LoRa, helps manage larger networks through gateways and cloud platforms. **Figure 1** shows an illustrative high-level overview of a typical IoT network architecture.

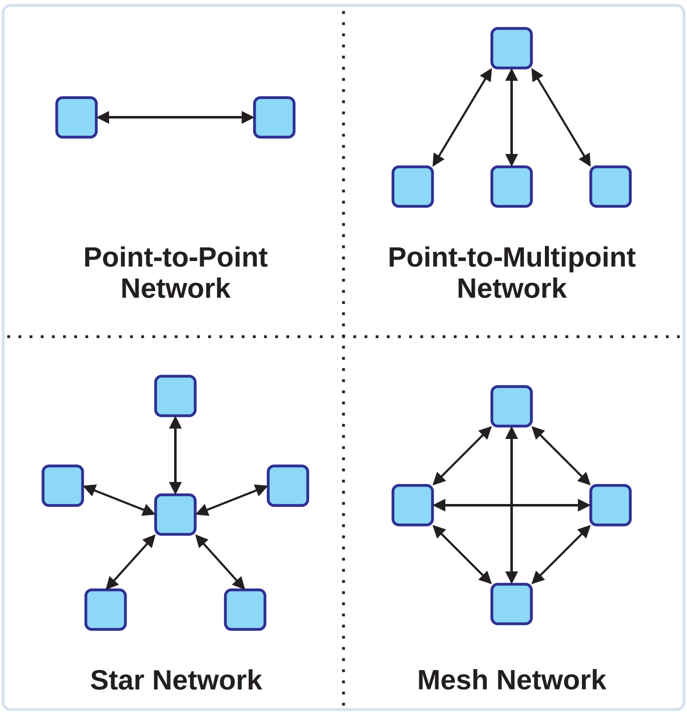


Figure 2: Types of Wireless communication topologies.

It highlights how multiple wireless protocols, including Wi-Fi, Zigbee, LoRa, and LTE, can individually connect sensors and end devices to the internet. This simplified representation demonstrates potential use cases and provides clarity on how different wireless technologies can fit into a unified IoT ecosystem.

In standard LoRaWAN networks, end nodes do not directly communicate with each other; instead, they send data exclusively to centralized gateways. This star-of-stars topology simplifies network management (**Figure 2**), greatly reduces node complexity, and significantly extends battery life. However, custom LoRa implementations (outside LoRaWAN) can support direct node-to-node or mesh communication, useful in specific scenarios requiring decentralized communication.

LoRaWAN networks do have specific regulatory and practical transmission limitations, primarily determined by regional duty cycle restrictions in unlicensed frequency bands. For instance, in Europe (868 MHz), the duty cycle limit is typically 1%, meaning a device may transmit only about 36 seconds per hour. Networks like The Things Network (TTN) further recommend a fair-use policy of around 30 seconds of airtime per day per device, equating to roughly 10 to 20 small payload messages daily. These constraints help ensure efficient spectrum usage, minimize interference, and maintain network reliability.

The main advantages of LoRa are its long range and battery-friendly nature. Its disadvantages include very low data throughput, significant latency, and complexity in network management for large installations. LoRa is ideal for remote environmental monitoring, agriculture, utility metering, and any application involving scattered, long-range, battery-operated sensors.

Sigfox is a proprietary Low-Power Wide-Area Network (LPWAN) technology tailored for low-data-rate, long-distance IoT applications. It operates in sub-GHz unlicensed frequency bands (868 MHz in Europe and 902 to 928 MHz in North America), using ultra-narrowband modulation, resulting in extremely low power consumption and extended battery life — often several years per device. Sigfox devices can send up to 140 messages per day, with each message limited to 12 Bytes, totaling a maximum of 1,680 Bytes daily. Due to these strict limitations, Sigfox is ideal for applications requiring infrequent and small data transmissions, such as asset tracking, remote sensors, utility metering, and environmental monitoring, but is unsuitable for real-time, high-throughput, or highly interactive communication.

Both Sigfox and LoRaWAN are LPWAN technologies designed for low-data-rate, long-range IoT applications, offering similar advantages such as extended battery life, long-range communication, and minimal power usage. However, significant differences set them apart. Sigfox is a proprietary, operator-managed network using ultra-narrowband modulation with strict daily limits and minimal downlink capability. In contrast, LoRaWAN is an open standard utilizing spread-spectrum modulation, offering flexible payload sizes and greater user control over gateways and network infrastructure. LoRaWAN also provides more robust security through mandatory AES-128 encryption, whereas Sigfox's security relies on user-managed encryption.

ZigBee (IEEE 802.15.4)

ZigBee is a wireless standard designed specifically for low-power mesh networks. It operates at 2.4 GHz, providing modest data rates around 250 kbit/s, ideal for simple data exchanges like sensor readings or command signals. ZigBee modules, such as XBee [9], TI CC2530, CC2652, Espressif ESP32-C5/C6 and Silicon Labs EFR32, support mesh topologies allowing networks to expand coverage across large buildings or campuses through multiple hops.

The range per hop typically varies from 10 to 100 m depending on environment and antenna quality. ZigBee devices are highly energy-efficient, often running for several years on batteries. The main advantages are low power, robust mesh capability, and reliable network coverage. However, the complexity of ZigBee network setup, limited throughput, and lack of direct IP connectivity without gateways can be disadvantages. ZigBee is widely adopted in smart homes, building automation, and industrial sensor networks.

ZigBee utilizes a mesh network topology, allowing nodes to communicate directly and relay messages through routers to extend coverage and reliability. This contrasts with LoRaWAN's simpler star-of-stars topology, which restricts end nodes to direct communication only with gateways. ZigBee's mesh architecture makes it highly suitable for applications requiring robust, reliable local coverage, such as home automation and building management systems.

ESP-NOW

ESP-NOW is a proprietary peer-to-peer wireless communication protocol developed by Espressif [10] specifically for ESP32 and ESP8266 devices. It enables fast, direct communication without the need for a Wi-Fi network or router. Operating at 2.4 GHz, ESP-NOW supports data rates up to around 2 Mbit/s with extremely low latency (less than 10 ms). It has a typical range of 30 to 300 m, depending on antenna and environment.

ESP-NOW devices consume relatively low power compared to standard Wi-Fi connections, especially when used in sleep modes. Typical hardware includes standard ESP32 and ESP8266 development boards. ESP-NOW's main advantages are ease of implementation, low latency due to the missing overhead of the Wi-Fi protocol, low power consumption, and quick device pairing. The primary disadvantage is that it is limited to Espressif hardware, does not support IP networks directly, and has limited payload size (~250 bytes). It works best for closed, device-to-device communication scenarios such as remote controls, telemetry, and sensor-actuator pairs.

Nordic RF (nRF24L01+)

Nordic's nRF24L01+ modules [11] are popular, low-cost, low-power RF transceivers operating at 2.4 GHz. They support (nominal) data rates of 250 kbit/s, 1 Mbit/s, and 2 Mbit/s, and provide simple, short-range communication typically up to 100 m in open areas, and around 30 to 50 m indoors.

Typical hardware usage involves pairing with microcontrollers boards from Arduino, STM32, and ESP32. These modules consume very low power, especially when used in sleep modes, enabling battery-operated designs. Their primary advantages are affordability, ease of use, very low latency, good documentation, and flexibility in simple point-to-point or star network setups. Disadvantages include lack of IP support, absence of native mesh capability, and manual handling of addressing and retries in firmware. nRF24L01+ is best used in DIY projects, simple telemetry systems, remote controls, and basic wireless sensor networks.

The nRF24L01+ transceiver module, once a staple in wireless communication for embedded systems, is now considered obsolete for new designs. Nordic Semiconductor has officially marked the nRF24 series as "Not recommended for new designs," encouraging developers to transition to more modern alternatives like the nRF52, and nRF54L series. Despite its obsolescence status, the nRF24L01+ remains available through various distributors, including Digi-Key and Mouser, making it accessible for legacy projects or educational purposes.

LTE (Long Term Evolution/4G)

LTE, often called 4G, is a cellular communication standard that provides high-speed wireless data connections over licensed frequency bands (typically 700 MHz to 2600 MHz). Data rates for embedded LTE modules typically range from several Mbps up to around 100 Mbit/s (Cat 4 and higher). Typical hardware includes modules like SIMCom SIM7600 and Quectel EC25 [12]. LTE consumes moderate to high power, making it challenging for battery-powered sensors unless carefully managed using power-saving modes like eDRX and PSM. LTE's range depends on the cell network but can extend several kilometers, making it ideal for mobile, automotive, tracking, and remote IoT devices needing internet connectivity over large distances. The main advantages of LTE are excellent coverage, reliable connectivity, and high data throughput. However, LTE has higher costs due to hardware complexity, SIM subscriptions, and power usage.

GSM (2G/3G)

GSM (Global System for Mobile Communications) is an older cellular standard, typically operating at frequencies around 850/900/1800/1900 MHz. Data rates for GSM (2G) are limited to around 10 to 200 kbit/s, suitable only for simple telemetry. Typical GSM modules like SIM800 or SIM900 [13] are inexpensive and well-supported by embedded systems. Power consumption is moderate, typically higher than modern LPWAN technologies but lower than LTE. Range is cellular-dependent, extending multiple kilometers. GSM's main advantage is very broad coverage and affordability. Disadvantages include very low data throughput, increasing network shutdown globally, and limited support going forward. GSM is best for legacy remote sensor systems, SMS-based telemetry, or where minimal data transmission is required.

NB-IoT (Narrowband IoT)

NB-IoT is a cellular-based low-power wide-area (LPWA) technology specifically designed for IoT applications. It uses licensed LTE bands, offering robust and reliable connectivity. Data rates range from 20 to 250 kbit/s, suitable for small payloads. NB-IoT modules like SIMCom SIM7020, Quectel BC95 [14], and Nordic nRF9160 [15] are increasingly popular. The technology is extremely power-efficient, enabling years of battery operation when using power-saving modes. NB-IoT supports extensive coverage (up to several kilometers indoors or underground due to high link budget) and can handle thousands of devices per base station. Its primary advantages include excellent battery life, deep indoor penetration, secure communication, and broad cellular coverage. Disadvantages are low throughput, higher latency (seconds to minutes), and dependence on cellular providers. Ideal use cases are metering, asset tracking, agriculture, and smart city deployments.

Learn more about LoRa, Wi-Fi and other wireless technologies

Want to get some more background about BLE or Sigfox? Build a beginners Wi-Fi project with ESP32 and the Arduino IDE? Or dive into the secrets of sensor networks with LoRa? The Elektor archive hosts hundreds of good Wireless projects and trainings, articles, videos and webinars. Ask Elektor GPT for the article/video which fits best for your needs!
www.elektormagazine.com/gpt

Comparison Table

Protocol	Data Rate*	Range (Typical)	Power Consumption	Hardware Examples	Topology	Suitable Environment	Best Use Cases
Wi-Fi	10 to 100+ Mbit/s	30 to 100 m (indoor/outdoor)	High	ESP32, ESP8266, Raspberry Pi	Star	Indoor, power-rich environments	Streaming, cloud IoT hubs
BLE	~1 Mbit/s	5 to 30 m (indoor/LOS)	Very Low	ESP32, Nordic nRF52, HM-10	Star	Indoor short-range applications	Wearables, mobile interfaces
ZigBee	250 kbit/s	10 to 100 m (per node, mesh)	Low	XBee, TI CC2530, CC2652, EFR32	Mesh	Indoor, multi-node systems	Smart homes, building automation
LoRa	300 bit/s to 50 kbit/s	2 to 10 km (outdoor)	Very Low	SX1276, RFM95, Heltec LoRa boards	Star (gateway)	Outdoor, remote locations	Agriculture, remote environmental sensing
ESP-NOW	2 to ~50 Mbit/s	100 to 300+ m, upto 1 km in LR Mode	Low	ESP32, ESP8266 modules	Peer-to-peer	Small ESP ecosystems	Remote controls, fast local telemetry
Nordic RF	250 kbit/s to 2 Mbit/s	30 to 100 m	Low	nRF24L01+ modules, Arduino	Star/Peer-to-peer	Basic DIY setups, indoor or outdoor	Wireless sensors, DIY telemetry
LTE (4G)	10 to 100 Mbit/s	Several km (cellular coverage)	Moderate to High	Quectel EC25, SIM7600, u-blox SARA	Star (cellular)	Outdoor, urban, suburban	Automotive, remote monitoring, mobile IoT
GSM (2G)	10 to 200 kbit/s	Several km (cellular coverage)	Moderate	SIM800, SIM900	Star (cellular)	Rural, remote telemetry	Legacy telemetry, SMS alerts, simple tracking
NB-IoT	20 to 250 kbit/s	Several km (cellular coverage)	Very Low	SIM7020, Quectel BC95, nRF9160	Star (cellular)	Deep indoor, remote outdoor locations	Smart meters, agriculture, deep indoor IoT
LTE-M	~1 Mbit/s	Several km (cellular coverage)	Low to Moderate	SIM7000, Quectel BG96, u-blox SARA-R4	Star (cellular)	Urban, suburban, mobile	Wearables, asset tracking, industrial sensors
5G	100 Mbit/s to 1+ Gbit/s	Dense urban environment (cellular coverage)	High	Quectel RG500Q, SIM8200 series, Fibocom modules	Star (cellular)	Urban environments, industrial	High-speed industrial, video, automotive, AR/VR

** Data rates are nominal, practical data rates can be far below these values.*

LTE-M (LTE Cat-M1)

LTE-M (or LTE Cat-M1) is another low-power, wide-area cellular technology tailored to IoT. Operating within standard LTE bands, LTE-M provides better data throughput than NB-IoT (typically around 1 Mbit/s maximum) and lower latency (hundreds of milliseconds). LTE-M modules like SIM7000 [16], Quectel BG96 [17], and u-blox SARA-R4 [18] are widely available. Power consumption is very low in sleep modes, offering multi-year battery operation. LTE-M supports mobility and voice services better than NB-IoT, making it suitable for applications requiring intermittent data with moderate payloads and occasional mobility. Its advantages include better latency and data rates than NB-IoT, power efficiency, and seamless integration with existing LTE infrastructure. Drawbacks include slightly higher power usage compared to NB-IoT and reliance on cellular networks with

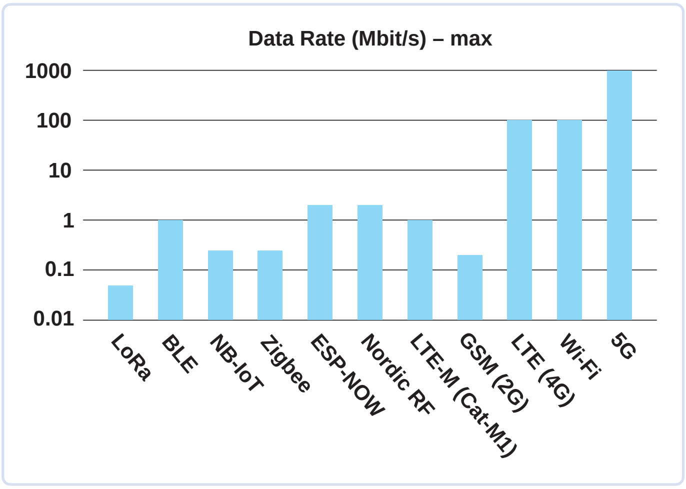


Figure 3: Comparison of Maximum Data Rates Across Wireless Technologies.

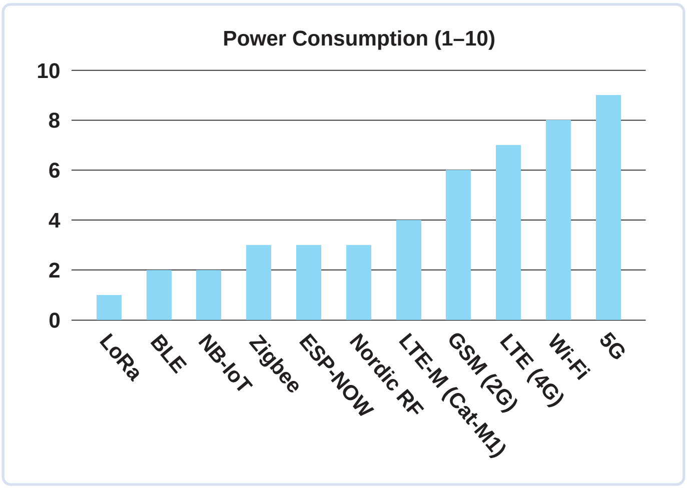


Figure 4: Relative Power Consumption Scale for Different Wireless Technologies (1 = lowest, 10 = highest).

variable global coverage. LTE-M works best in wearables, asset tracking, smart meters, and remote sensors with moderate data needs.

5G

5G (Fifth Generation) is the newest cellular standard, offering significantly higher throughput (gigabit speeds), extremely low latency (1 to 10 ms), and enhanced network reliability and capacity. 5G operates in various frequency bands, including Sub-6 GHz and mmWave (above 24 GHz). Embedded 5G modules (e.g., Quectel RG500Q [19]) are just entering mainstream use but are expensive and consume significant power, limiting their suitability for simple embedded devices or sensors. Its main advantages include massive bandwidth, ultra-low latency, and support for extremely dense device deployments. The drawbacks are high costs, high power consumption, and limited global network deployment currently. 5G is ideal for high-speed industrial automation, real-time video streaming, automotive applications, augmented reality, and any future high-bandwidth embedded solutions.

Choosing the Right Wireless Technology

Figure 3 through Figure 6 provide a clear comparative overview of key performance metrics. **Figure 3** highlights the substantial differences in maximum data rates, illustrating Wi-Fi and 5G as optimal choices for high-throughput applications, while LoRa and BLE support significantly lower bandwidth, suitable for simpler sensors. In **Figure 4**, the relative power consumption ratings indicate LoRa, BLE, and Nordic RF as superior for battery-powered scenarios, contrasting sharply with the higher energy demands of Wi-Fi and 5G. **Figure 5** outlines the range capabilities of LoRa and cellular technologies like LTE-M and NB-IoT, compared to limited short-range solutions like BLE and Zigbee.

Finally, **Figure 6** compares the minimum achievable latencies, emphasizing 5G and Wi-Fi for low-latency applications, whereas LoRa and NB-IoT show significantly higher latencies, appropriate for less time-sensitive deployments. Note that Figures 3, 5, and 6 use logarithmic scales on the vertical axes to effectively display

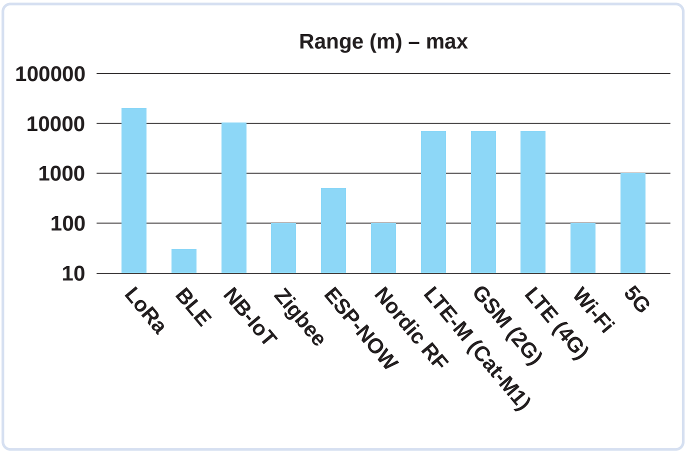


Figure 5: Comparison of Maximum Communication Range for Various Wireless Protocols.

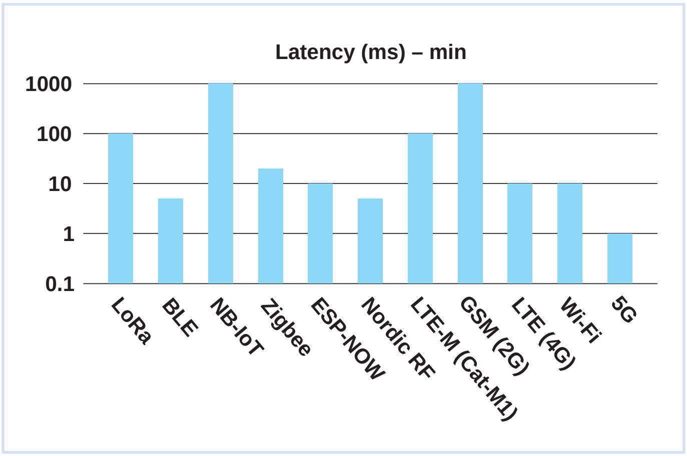


Figure 6: Minimum Achievable Latency Comparison Among Common Wireless Technologies.

the wide performance variations among these technologies. Ultimately, understanding these trade-offs will help you choose the most suitable wireless solution tailored to your project's specific requirements.

250432-01

About the Author

Saad Imtiaz, Senior Engineer at Elektor, is a mechatronics engineer who has extensive experience in embedded systems and product development. His journey has seen him collaborate with a diverse array of companies, from innovative startups to established global enterprises, driving forward-thinking prototyping and development projects. With a rich background that includes a stint in the aviation industry and leadership of a technology startup, Saad brings a unique blend of technical expertise and entrepreneurial spirit to his role at Elektor. Here, he contributes to project development in both software and hardware.

Questions or Comments?

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

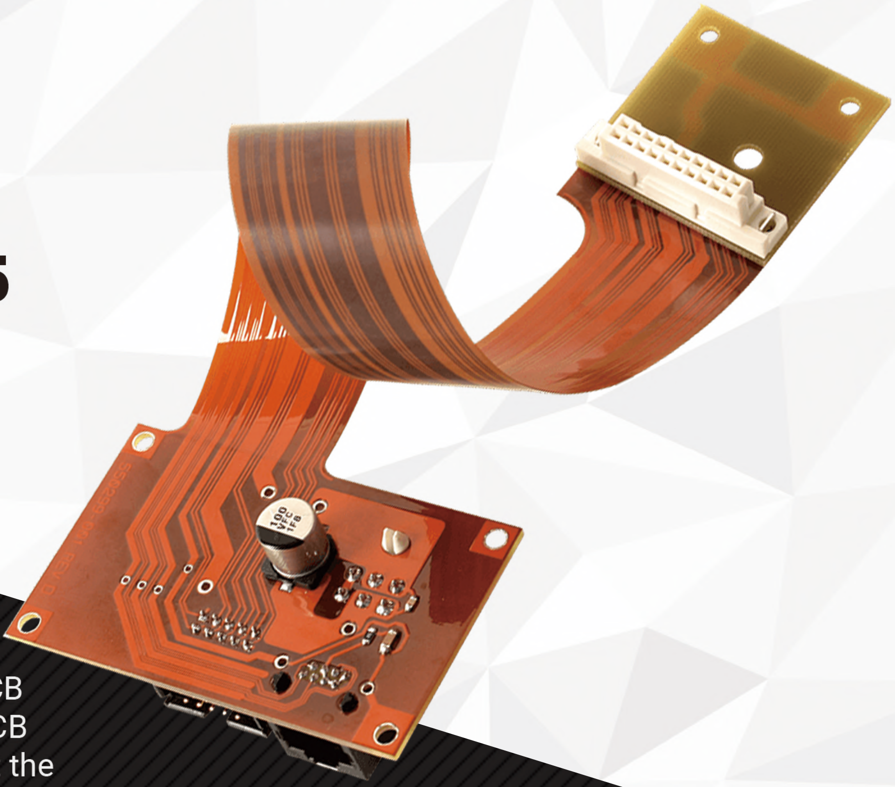
WEB LINKS

[1] Espressif ESP32-C5: https://www.espressif.com/en/products/socs/esp32-c5
[2] Nordic Semiconductor – nRF52 Series (BLE): https://www.nordicsemi.com/Products/Wireless/Bluetooth-Low-Energy
[3] Semtech – SX1276 LoRa Transceiver: https://www.semtech.com/products/wireless-rf/lora-connect/sx1276
[4] Microchip – RN2483 LoRa Module: https://www.microchip.com/en-us/product/RN2483
[5] STMicroelectronics - LoRaWAN Products: https://www.st.com/en/wireless-connectivity/lorawan-products.html
[6] Seeed Studio – Wio-E5 LoRa Module: https://www.seeedstudio.com/LoRa-E5-Wireless-Module-p-4745.html
[7] Ebyte – LoRa Modules: https://www.cdebyte.com/Module-Lora
[8] RAK Wireless – LoRaWAN Modules: https://store.rakwireless.com/collections/wisduo-modules-for-lorawan
[9] Digi International – XBee ZigBee Modules:
https://www.digi.com/products/embedded-systems/digi-xbee/rf-modules/2-4-ghz-rf-modules/xbee-zigbee
[10] Espressif Systems – ESP-NOW: https://www.espressif.com/en/solutions/low-power-solutions/esp-now
[11] Nordic Semiconductor – nRF24L01+ Module: https://www.nordicsemi.com/Products/nRF24-series
[12] Quectel – EC25 LTE Module: https://www.quectel.com/product/lte-ec25-series/
[13] SIMCom – SIM800 GSM Module: https://www.simcom.com/module.html
[14] Quectel – BC95 NB-IoT Module: https://www.quectel.com/product/lpwa-bc95-gr/
[15] Nordic Semiconductor – nRF9160 NB-IoT Module: https://www.nordicsemi.com/Products/nRF9160
[16] Botletics - SIM7000 LTE-M Module: https://www.botletics.com/products/sim7000-shield
[17] Quectel – BG96 LTE-M/NB-IoT Module: https://www.quectel.com/product/lpwa-bg96-cat-m1-nb1-egprs/
[18] Trasna/u-blox – SARA-R4 LTE-M/NB-IoT Module:
https://www.trasna.io/products/sara-r42-cellular-iot-module?field_file_lifecycle=swdesign
[19] Quectel – RG500Q 5G Module: https://www.quectel.com/5g-iot-modules/

PCBWay

UNLEASH INNOVATION WITH FLEX & RIGID-FLEX PCB SOLUTIONS

Meet PCBWay at
E&A en LabNL 2025
Booth 9A-011



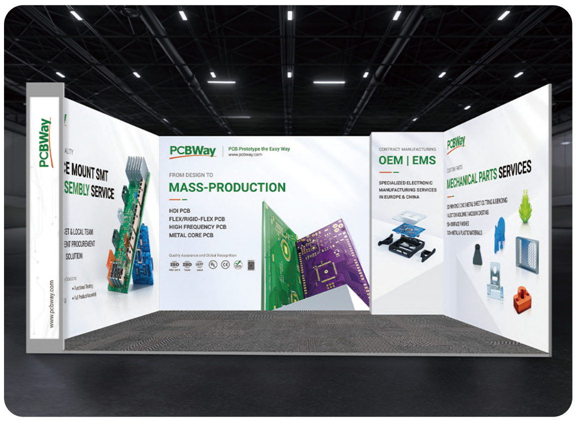
At PCBWay, we push the boundaries of PCB technology with our Flex and Rigid-Flex PCB manufacturing services, designed to meet the demands of today's compact, high-performance electronics.

HIGH DURABILITY & RELIABILITY
Ideal for dynamic and high-density applications.

SPACE-SAVING DESIGN
Optimize your product footprint with flexible interconnections.

ADVANCED MATERIALS & PROCESSES
Tailored solutions for aerospace, medical, automotive, and wearable electronics.

Whether you're in prototyping or mass production, our one-stop solution ensures precision, fast turnaround, and unmatched quality control.



Visit us at Booth 9A-011
E&A en LabNL 2025

Discover how PCBWay can accelerate your next project with cutting-edge Flex and Rigid-Flex PCB solutions.

LET'S TURN YOUR BOLDEST IDEAS INTO REALITY.

www.pcbway.com
service@pcbway.com

Satellite Tracking Using LoRa

The TinyGS Network Bringing Space Data to Earth

By Bryce Vandegrift (United States of America)

With the explosion of small satellites launched by universities, hobbyists, and private groups, tracking them has become a major challenge. TinyGS offers a compelling solution: an open-source, community-powered global network that receives satellite telemetry using inexpensive LoRa hardware. By leveraging the long range, low power, and interference-resistant nature of LoRa radios, TinyGS enables anyone to set up a ground station with consumer-grade components and start collecting satellite data from their own backyard.

The amount of small satellites currently in orbit can be quite astounding. From universities to individuals, it seems that anyone can put a proper satellite into space, given enough effort. One problem that arises from this though is actually tracking those satellites, that's where TinyGS steps in. TinyGS is a community run open-source global satellite tracking network that uses LoRa technology.

Satellite tracking and data collection can be quite a tricky endeavor with the sheer number of satellites in orbit potentially operating on different frequencies. Luckily, many of these satellites have a LoRa modem on them to send telemetry data and small bits of information. The great thing about LoRa (which is short for long-range) is, as the name implies, the fact that it can send data over very long ranges. How long of a range can these radios achieve? In some cases, over 1,300 km (830 miles) on land! That makes LoRa perfect for low bandwidth transmissions on satellites.

TinyGS leverages the LoRa radios on these satellites to effectively track them and receive data from them using readily available consumer-grade hardware. The information collected by TinyGS is not limited to just satellite positional data, but sometimes also includes information about the status of the satellite itself, information about surrounding satellites, and more. TinyGS makes all the information received from these satellite public to contribute to the future of space exploration openly.

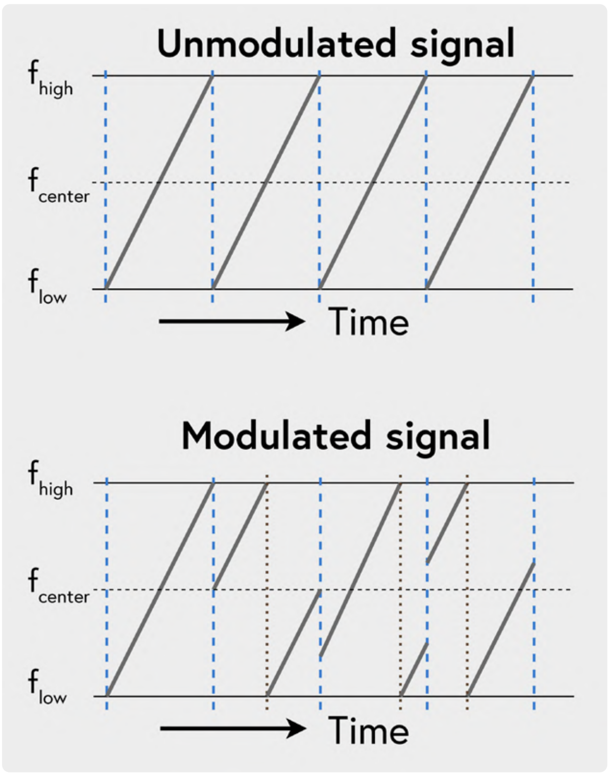


Figure 1: A example showing a LoRa "up-chirp" signal being modulated. Source: lora.readthedocs.io.

Why LoRa?

Some might ask: Why use LoRa on satellites? Although there are many reasons why someone might want a LoRa radio on their satellite, it usually boils down to range, cost, power consumption, and interference. We mentioned above how LoRa can achieve ranges of over 1,000 kilometers. LoRa radios also happen to be somewhat simple and inexpensive. An average consumer LoRa module can cost around $5, which isn't too costly compared to other long-range communication modules. LoRa modems are also extremely energy efficient and, depending on the specific application, a battery-powered project using a LoRa modem can last years on a single charge!

However, one of the biggest selling points for the use of LoRa in satellites is its resistance to interference. For those unfamiliar with how the LoRa protocol works, LoRa uses a modulation technique called Chirp Spread Spectrum (CSS) to transmit and receive data. CSS uses a wide frequency range to send "chirps," which are signals that increase or decrease in frequency over time. The transmission data is then encoded or modulated into these "chirps" and sent over the air. This method is very power efficient, but most importantly, it's extremely resistant to interference. An illustration of how a LoRa "chirp" is used can be found in **Figure 1**. When it comes to satellite communications, one of the biggest challenges is accounting for and mitigating the Doppler effect in fast-moving objects such as aircraft and satellites. Because of the LoRa protocol's frequency changing "chirps," it actually makes LoRa signals extremely resilient against the Doppler effect, more so than most alternatives out there.

With a combination of low cost, low power-consumption, long range, and very low interference, it's no wonder why many satellites have at least one LoRa modem attached. These features of the LoRa protocol are also why TinyGS takes advantage of the LoRa modems on satellites, since it makes it easier for someone with limited resources to participate in the TinyGS network.

How Does the TinyGS Network Work?

Operation of the TinyGS network itself is actually quite simple. Every TinyGS ground station sends any information that it collects to an MQTT server. For those unfamiliar with MQTT, it is a simple protocol that is lossless and light on bandwidth for sending messages and telemetry over limited bandwidth connects. Since the MQTT protocol is very lightweight, it is perfect for systems with limited resources, such as microcontrollers. The MQTT server will then forward any received data, which may optionally be decoded and stored in a database.

Once the data is received and stored in the TinyGS servers, it is made publicly available to anyone and everyone who wants it. Received packets and ground station status are regularly updated in the TinyGS Telegram channel using a custom-built Telegram bot. Obviously, providing all TinyGS data over Telegram would be quite impractical, to say the least. Luckily, TinyGS provides a REST API for programmers and anyone else who wants to grab ground-station data. For non-programmers, the same information is also provided on the TinyGS website [1] in a more human-readable format. An overview of the TinyGS network can be seen in **Figure 2**.

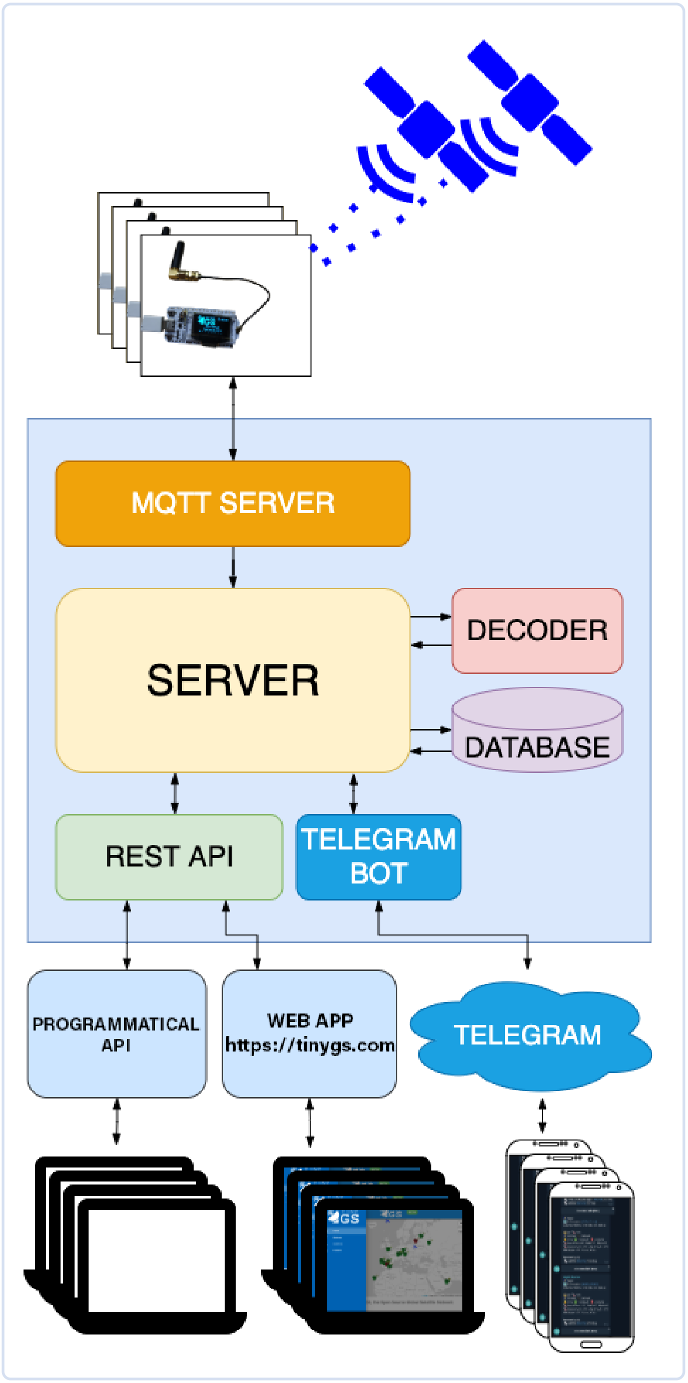


Figure 2: Architecture overview of the TinyGS network. Source: https://github.com/G4lile0/tinyGS.

Required Hardware

Setting up a LoRa ground station for use in the TinyGS network is easy and simple. The TinyGS firmware officially supports ESP32 modules that have the sx126x or sx127x series LoRa modules on board. The TinyGS project has a list of officially supported ESP32 LoRa boards, which you can find at [2], or you can build your own using the *Board Template* feature built into the TinyGS firmware.

One important thing to note when buying or making a board is the LoRa modem frequency. Most LoRa modems have two preset frequency ranges: 433 MHz and 868 to 915 MHz. Most LoRa satellites currently in orbit use the 433 MHz range, since it has a longer propagation and is less prone to interference. However, ground stations using either

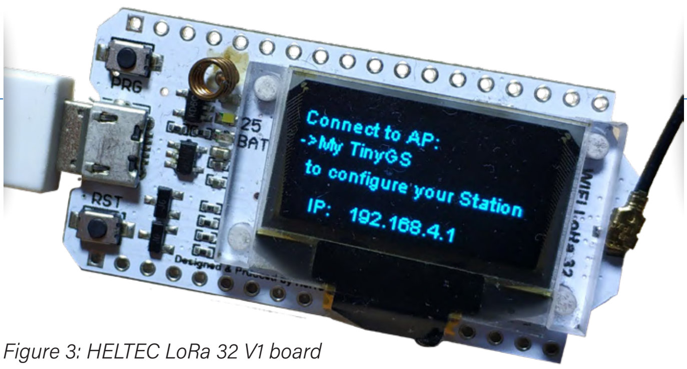


Figure 3: HELTEC LoRa 32 V1 board running the TinyGS firmware.

frequency range are always welcome on the TinyGS network. There is also the 137 MHz frequency range, which is used by just a few satellites, in addition to the SpaceX Starlink satellites, however (to my knowledge) there are no officially supported 137 MHz LoRa modems for TinyGS; contributions and development are welcome, though.

With a LoRa development board in hand, you'll probably also need a proper antenna, as the stock antenna that comes with most ESP32 LoRa boards is rather small and sad. You can buy or easily make your own antenna for whichever frequency range you choose. For making your own LoRa antenna easily and cheaply, the TinyGS project has many resources, which you can find at [3]. Just make sure that the antenna that you buy or make matches the frequency of the selected LoRa radio frequency.

Setting Up the Ground Station

Before we set the board up, we first need to grab valid credentials in order for our ground station to send data to the TinyGS network. TinyGS uses a Telegram bot to distribute MQTT credentials for their server. Although this method is quite unorthodox, it works pretty well and prevents spam and/or malicious data from being sent to their server. If you don't have a Telegram account, you will need to make one to connect to the TinyGS Telegram channel [4]. There, you can start a private chat with the bot using the link at [5]. All you need to do is send the command "/mqtt" to the bot and it will give you an MQTT username and password. Make sure to keep this handy when we're configuring our ground station, as these credentials are required to connect to the TinyGS network.

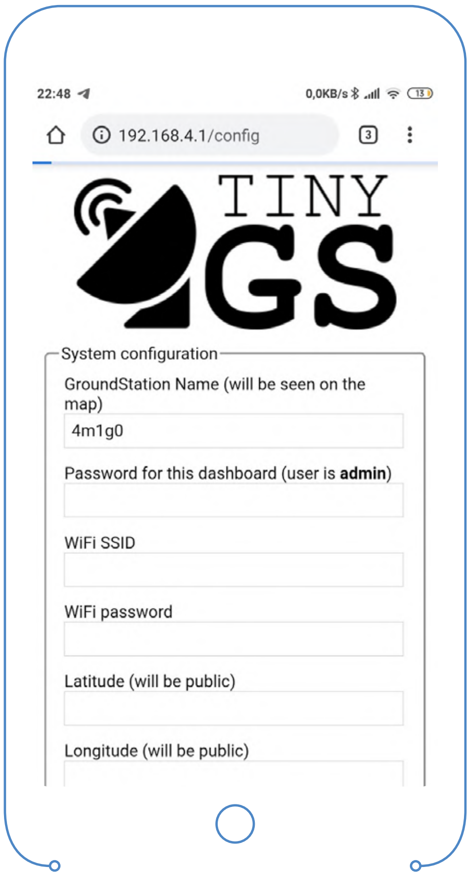
With our LoRa board in hand, we first need to flash the TinyGS firmware. To flash the firmware, TinyGS actually provides a new web-based installer at [6] for Chromium-based web browsers. Alternatively, you can flash the TinyGS firmware using PlatformIO, the Arduino IDE, or one of various other valid methods. An example of a fully powered board with the TinyGS firmware installed can be seen in **Figure 3**.

Once we apply power to our board, it will generate a Wi-Fi access point with the name *My TinyGS*. You can use a phone or a computer to connect to the access point to configure all the vital settings for your TinyGS ground station. In order to access the settings you'll need to open a web browser and go to the webpage at *192.168.4.1/config*. The configuration page of your ground station should look similar to **Figure 4**. Once open, you can change some very vital settings. There are a few settings that should probably be changed/set to get your ground station into working order.

First, we need a unique ground station name, this can be changed in the *GroundStation Name* setting. We should also set the password for the configuration dashboard so that nobody can tamper with our settings. You will also need to provide your Wi-Fi SSID and password in order for your ESP32 to send the collected satellite data over the internet to the TinyGS server. The latitude, the longitude, and the timezone of your ground station should also be set in order to accurately track where your ground station is located. Now is also a good time to make sure that the *Board Type* setting is correctly set to your board, otherwise it might not work! Remember the MQTT username and password that we obtained from Telegram earlier? We will need to enter those credentials into the configuration page to actually send our satellite data to the TinyGS network. Automatic tuning can also be enabled in order for your ground station to more accurately tune to an upcoming satellite's frequency, but this feature is still experimental.

Now We Wait

Congratulations! Your very own ground station is now set up and ready to go! You can look at your station dashboard by opening a web browser and going to *192.168.4.1*, which will provide you with information about the status of your ground station, what packets you have received, and your LoRa modem configuration. You can also find your ground station listed on the TinyGS website, and you can look at what satellites it has communicated with and what telemetry data it has received once a few satellites pass over it.


22:48
192.168.4.1/config

System configuration

GroundStation Name (will be seen on the map)
4m1g0

Password for this dashboard (user is **admin**)

WiFi SSID

WiFi password

Latitude (will be public)

Longitude (will be public)

Figure 4: Local TinyGS configuration web page.

It is quite a time to be alive for space exploration and development. Being able to freely interact with satellites was quite a dream for an individual as little as 20 years ago. Now, almost anyone can contribute to the advancement of space exploration and development without even leaving their backyard.

240668-01

About the Author

Bryce Vandegrift is an electronics hobbyist who likes designing, troubleshooting, and repairing electronics in his free time. He currently runs a personal blog and website at https://brycev.com and a YouTube channel on technology tutorials and system management at https://youtube.com/@brycevandegrift.

Questions or Comments?

Got any feedback about this article? Please contact the Elektor editorial team at editor@elektor.com.

Related Product

- **LILYGO T-Beam V1.2 ESP32 LoRa Development Board with 0.96" Display (EU868)**
  www.elektor.com/20590

WEB LINKS

[1] TinyGS: https://tinygs.com
[2] TinyGS hardware, GitHub: https://github.com/G4lile0/tinyGS/wiki#hardware
[3] TinyGS antenna specifications, GitHub: https://github.com/G4lile0/tinyGS/wiki/Antenna
[4] TinyGS Community group — Telegram invitation link: https://t.me/joinchat/DmYSElZahiJGwHX6jCzB3Q
[5] TinyGS Personal Bot on Telegram: https://t.me/tinygs_personal_bot
[6] TinyGS web installer: https://installer.tinygs.com/

Engineered for Clarity –
in Every Waveform and Every Touch

SDG3000X Series

Arbitrary Waveform Generator

- Output frequency: 80/160/200 MHz
- Arbitrary waveform length: 40 Mpts
- Vertical resolution: 16-bit
- Sampling rate: 1.2 GSa/s
- Display: 7" Touch screen

SIGLENT®

www.siglenteu.com
Info-eu@siglent.com

4G-Compatible SMS Remote Control

Remotely Control Your Equipment

By Bruno Clerc (France)

How can you remotely control home heating? Here's an SMS remote control system using the mobile network, featuring relay outputs and a temperature sensor. It's based on an ATmega328P and the Arduino environment! Follow the development journey of this home automation project, including the transition from 2G to 4G.

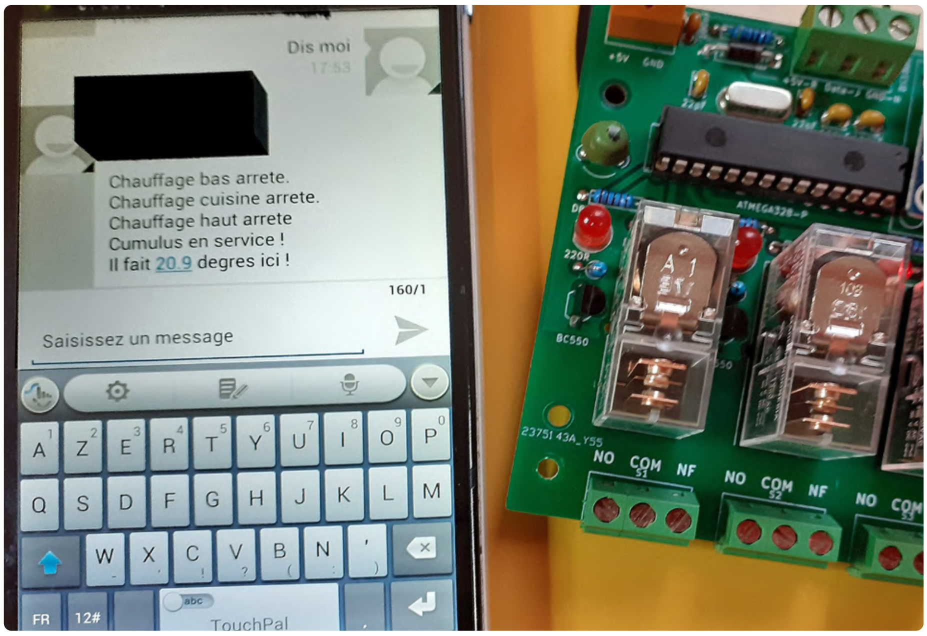

Dis moi
17:53
[redacted]
Chauffage bas arrete.
Chauffage cuisine arrete.
Chauffage haut arrete
Cumulus en service !
Il fait 20.9 degres ici !
160/1
Saisissez un message

Figure 1: The remote control receiver features relays controllable remotely via SMS.

Many of our readers who own a second home with electric heating will relate to this: the house is cold when you arrive. The same goes for hot water, which isn't available until the next day due to the long heating time of the water heater. Yet keeping these appliances running constantly when the house is unoccupied would be neither reasonable nor economical. Clearly, remote control would be an advantage.

Commercial solutions exist, and searching for "GSM remote switch" on AliExpress [1] yields several results. However, after testing one such device, I found it incompatible with French phone networks and poorly documented. Given the high cost of more reliable commercial options, I decided to build my own system. The first step was made easier thanks to an article [2] written by a radio amateur and electronics enthusiast, which helped me read SMS messages using an Arduino Nano and a SIM800 GSM module [3].

This project evolved over time and now supports 4G. My DIY solution allows me to use commands in French (you can, of course, adapt the code to your language), and includes a frost protection mode (see further down).

Requirements Specification

The primary objective is to control four relays by sending SMS commands from a mobile phone (**Figure 1**). In my setup, three relays control radiators, and the fourth manages the electric water heater. Relay states must be saved in EEPROM so that, in the event of a power outage, the heating controller resumes exactly as it was before the interruption.

Upon first startup, the receiver must be paired by sending an SMS with an activation code. This unique code registers the sender's phone number as the "master number." Only this master number can later configure access permissions and manage critical functions.

I also included the ability for the master to register two guest numbers, via SMS.

These guests have limited rights: they can activate or deactivate relays, but cannot change system settings. Another important feature is remote reset. In case the master number is lost or changed, I wanted a way to regain control. I added a reset code that, when sent from any phone, erases the configuration and restarts pairing from scratch.

Phone numbers are stored in EEPROM to preserve data over power cycles. To avoid working "blind," the system should reply with a complete status report when queried via SMS. Ideally, the remote control system should also return the stored guest numbers.

To integrate neatly with my electrical installation, I designed the PCB to fit inside a standard DIN rail enclosure. The 5 V power supply is also a modular DIN-mounted unit. If the power levels to be switched by the relays exceed their rated capacity, external DIN-mount contactors or relays can be used. In that case, the remote unit's relays only switch the coil current of those external devices.

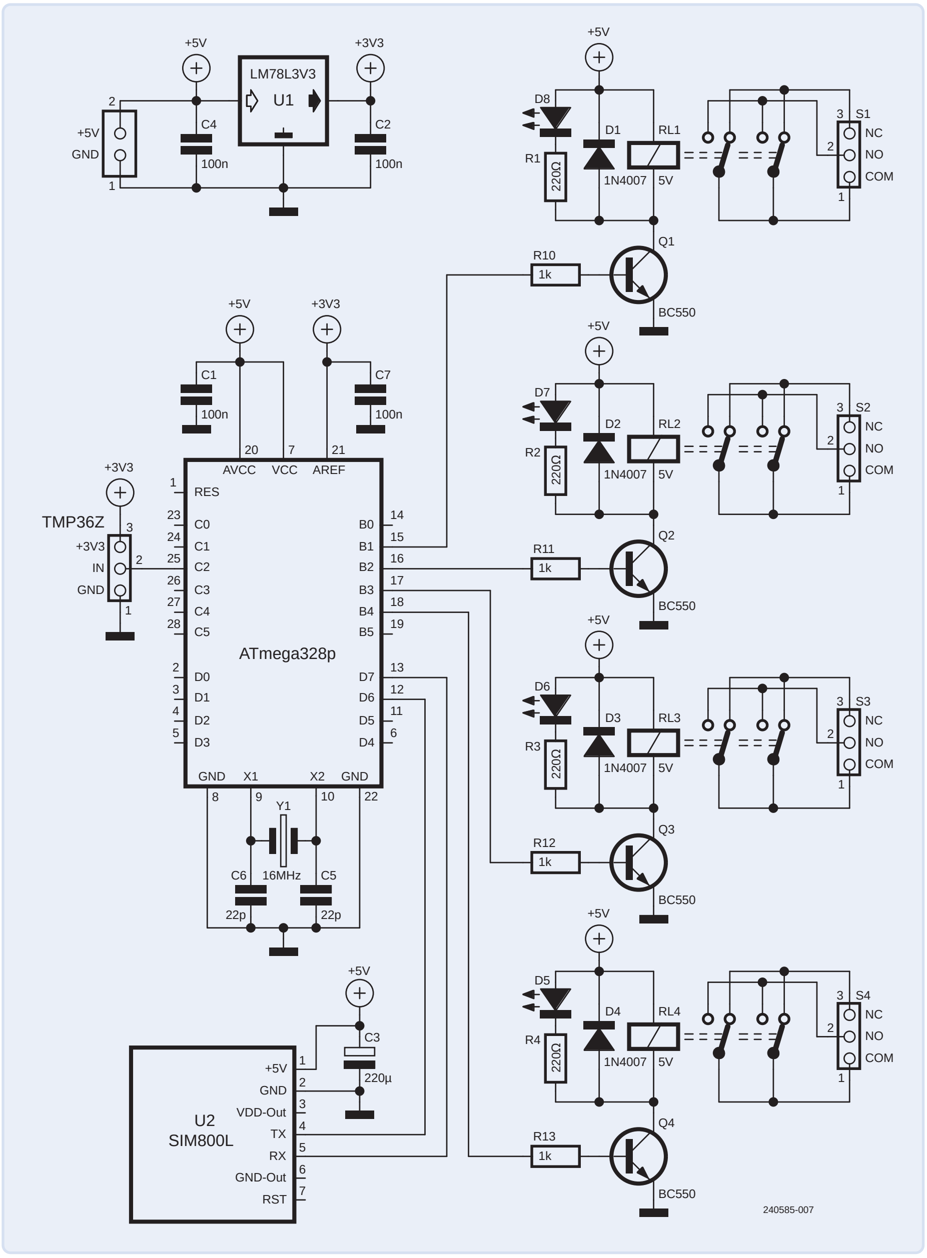


Figure 2: Schematic of the first version, based on the 2G SIM800 modem.

Frost Protection Mode

Electric radiators usually include a frost protection mode that maintains room temperature at a minimum of +7°C to prevent condensation and mold, while consuming minimal electricity. However, in my system, the radiators are either fully on or off via relays, and in the off state, they receive no power, so their built-in frost protection won't work. The receiver must therefore independently reproduce this behavior using its own ambient temperature sensor.

First Version: 1.1

A first version that met these requirements was completed in August 2022. The schematic is shown in **Figure 2**. It includes a PCB I designed myself, featuring a Microchip ATmega328P microcontroller programmed using the Arduino environment. It is accompanied by four DPDT relays, controlled via outputs PB1 to PB4 of the ATmega through four BC550 transistors. SMS reception is handled by a SIM800 GSM module, which communicates with the microcontroller through a serial interface. An LED in parallel with each relay indicates its state. As usual, freewheeling diodes are placed in parallel with each relay coil to protect the transistors from voltage spikes during switching. The PCB for this first version is shown in **Figure 3**.

DS18B20 Or Not

At the start of development, I planned to use a DS18B20 temperature sensor, known for its accuracy and robustness. Software integration is straightforward thanks to the *DallasTemperature* library [4]. However, this library is somewhat heavy for use with an ATmega328P. As an example, the *Alarm* demo program provided within the *DallasTemperature* library, although not very complex, already uses 7.8 kB out of the 32 kB of available program storage space on the ATmega328P, or 24%. The problem is, my own code is becoming quite large too. Indeed, the numerous character strings used for SMS processing

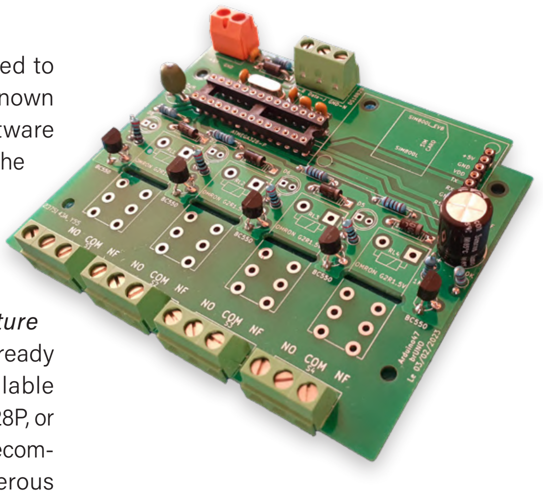


Figure 3: PCB of the first version.

take up significant space in this project. As a consequence, during code development, the compiler warned that I was nearing the flash memory limit.

A New Sensor

As a simple solution to this lack of storage space, I decided to change sensors. I chose the TMP36Z, a simple analog sensor that requires no dedicated library. Power it with 3.3 V and measure the output voltage, then use a simple formula to convert that voltage into a temperature reading. A TO-92 3.3 V regulator powers both the TMP36Z and the AREF pin of the microcontroller. This provides a stable reference voltage for analog conversions, ensuring reliable temperature readings. The only drawback is that it's not easy to build a clean and robust assembly at the end of the sensor cable, unlike with the DS18B20, which is commonly sold pre-encapsulated at the cable's tip. You can find this very first version of the project on Elektor Labs [5].

With the new sensor, I was able to reduce code size. Initially, the system in this first version was satisfactory, but occasional malfunctions began to appear. Sometimes, the receiver would remain unresponsive and failed to reply to SMS commands. Clearly, an update was needed.

Version 1.2

It turned out that the problem came from the SIM800 module, which sometimes lost its connection to the cellular network and couldn't reconnect automatically. I couldn't identify the exact cause, but I observed that pressing the module's reset button resolved the issue and allowed it to reconnect. In February 2023, a version 1.2 was created to address this [6]. I added Q5, an NPN BC550 transistor controlled by the ATmega328P's PB0 output, allowing the microcontroller to simulate a press of the SIM800's reset button, either at startup or when a network loss is detected.

While developing another project for Elektor [7], I explored much simpler methods to use a DS18B20 without relying on a heavy library. That prompted me to reuse a similar, compact code for version 1.2, enabling the use of a rugged encapsulated DS18B20 mounted at the end of a cable, for mechanical robustness as noted earlier. This time, the compiled code comfortably fits within the available memory, so the TMP36Z was replaced by a DS18B20 from then on. The DS18B20's data output is connected to input PC2 on the ATmega, with a 4.7-kΩ pull-up resistor. A 100-nF capacitor is used to decouple the power supply.

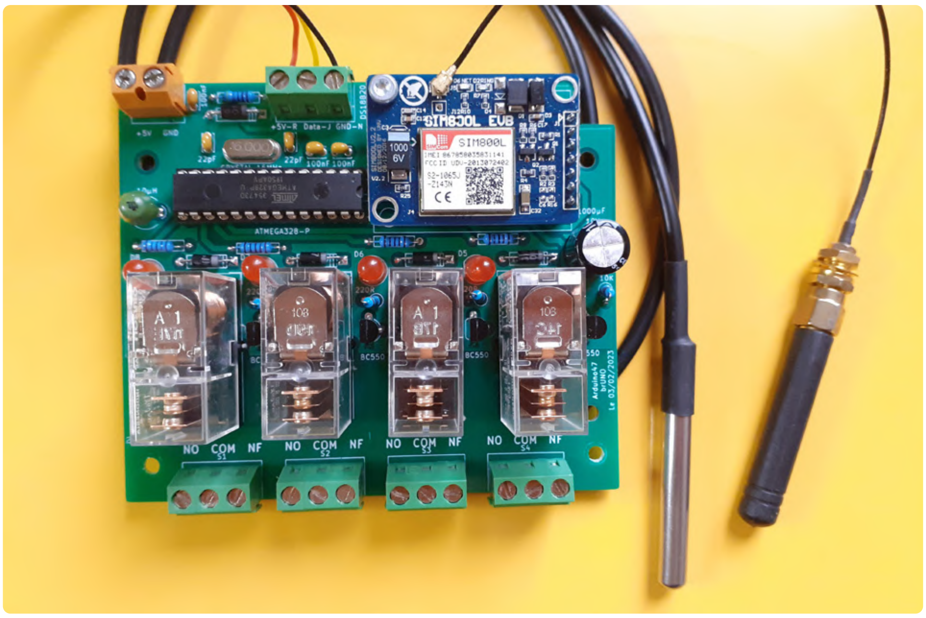


Figure 4: Version allowing remote modem reset, also featuring the more robust encapsulated sensor.

This version was also an opportunity to replace the initial 10-A Songle SRD series relays with more robust Omron G2R-1-E relays rated for 16 A. Additionally, a 10-µH inductor was added to the ATmega's 5 V power line to block interference from the modem. The complete assembly is shown in **Figure 4**.

The End of 2G Networks

With the gradual disappearance of 2G networks in France and other European countries, it became necessary to adapt the project. This time, there was no alternative but to replace the modem module. I chose the BK-A7670E (LTE/4G) module, which is a breakout board for the A7670E LTE modem [8], and decided to adapt it to the project while retaining the existing PCB.

In October 2024, I developed a UART adapter board that fits between the original PCB and the new modem module. Details are available on Elektor Labs [9]. The interface PCB is shown in **Figure 5**. The BK-A7670E is surface-mounted on the top side. On the underside, the male pins on the left allow the board to plug directly into the former SIM800 socket, with pin correspondence ensured.

This adapter allows for UART logic level conversion (5 V to 3.3 V) using BSS138 MOSFETs, and also enables modem reset via a BSR14 transistor. Additionally, the adapter provides the necessary 3.3-V power supply for the module, using a TLV76033DBZR regulator integrated directly into the PCB.

To make the new modem on its adapter PCB work with the existing mainboard PCB

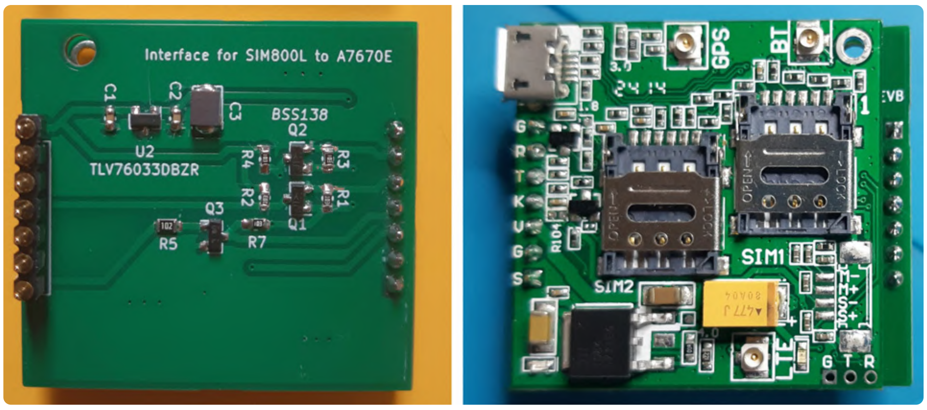


Figure 5: Adapter board used to upgrade to the 4G version.

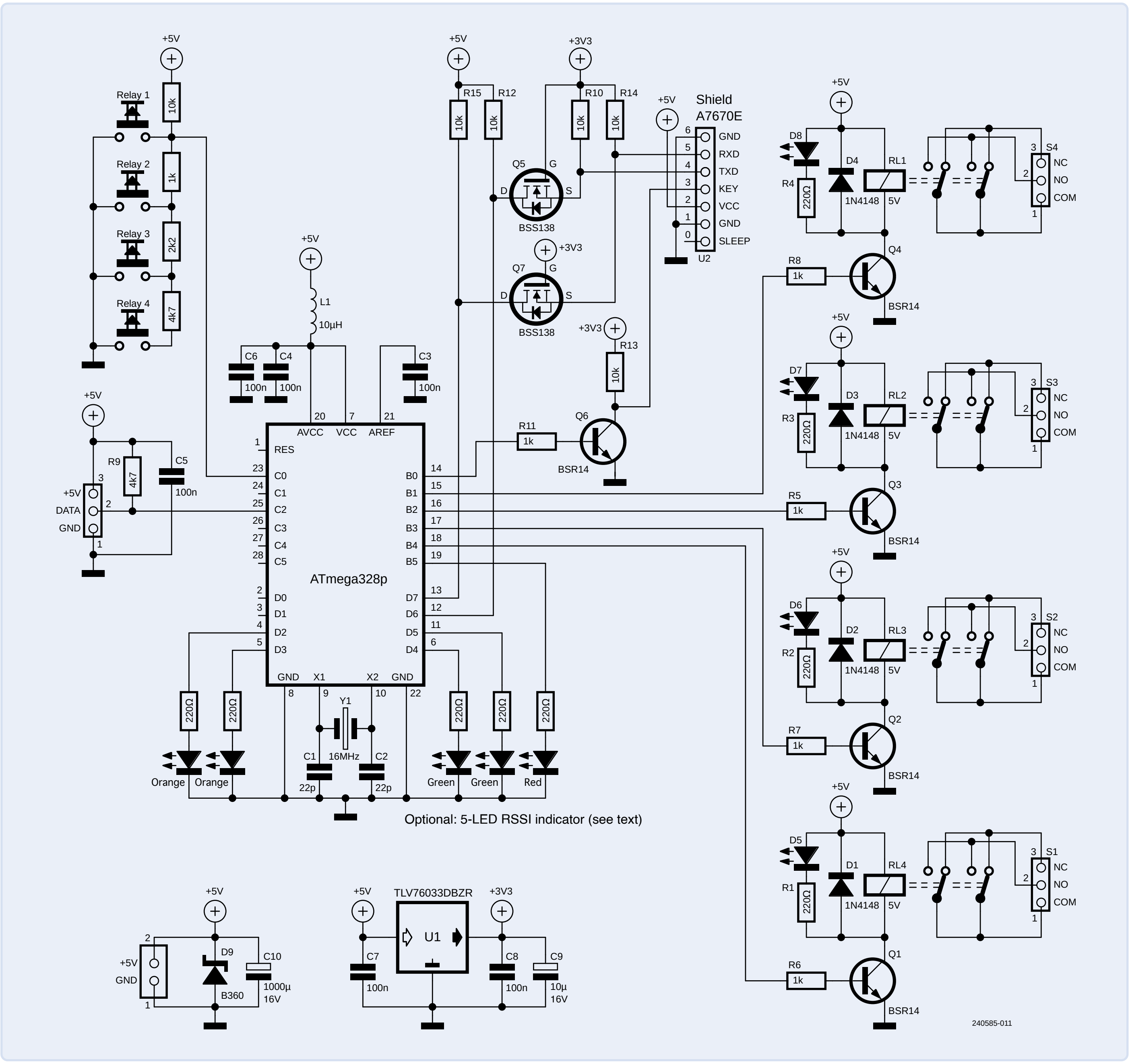


Figure 6: Schematic of the latest version, on a single PCB.

from the previous version, a few additional changes are required. On the SIM800L, the reset input has a built-in pull-up resistor and must be connected to ground to restart the module. In contrast, the BK-A7670E features a PWRKEY input with a pull-down resistor (R104); to simulate a button press and restart the module, this input must be pulled high. Consequently, components Q5, R5, and R6 must be removed. Also, the ATmega's PB0 pin must be connected to the RST_PIN_SIM800 pin of the new module. It's also important to note that, with my implementation of the on/off switch, the R104 resistor on the BK-A7670E shield must be removed to allow the ATmega to control the modem's power on/off. However, after removing R104, the modem won't boot by itself unless the microcontroller tells it so. This makes it difficult to change the modem's UART speed to 9600 baud, which is a required step covered hereafter. So, make sure to configure the modem first (see below) and remove R104 only after.

The Arduino code was, of course, updated with new AT commands matching the configuration and operation of the BK-A7670E 4G module.

Evolution: Everything on One PCB

Every electronics hobbyist wonders at some point: is a project ever really finished? The system worked flawlessly in 4G, but to give it a cleaner look, I decided to integrate everything onto a single PCB. Thus, version 2 was born in November 2024. Of course, instead of starting from scratch, I built on the existing design: the ATmega328P pins retain their original roles (controlling the four relays, reading the DS18B20 sensor, managing the 4G module, and UART), maintaining continuity in the project's logic. The complete schematic is shown in **Figure 6**, and the assembled PCB in **Figure 7**.

In this final version [10], the logic level converter is now integrated onto the board. The 4G module is simply soldered to the appropriate pin header. To enhance the system's versatility, this version now includes a local manual mode: the four relays can be

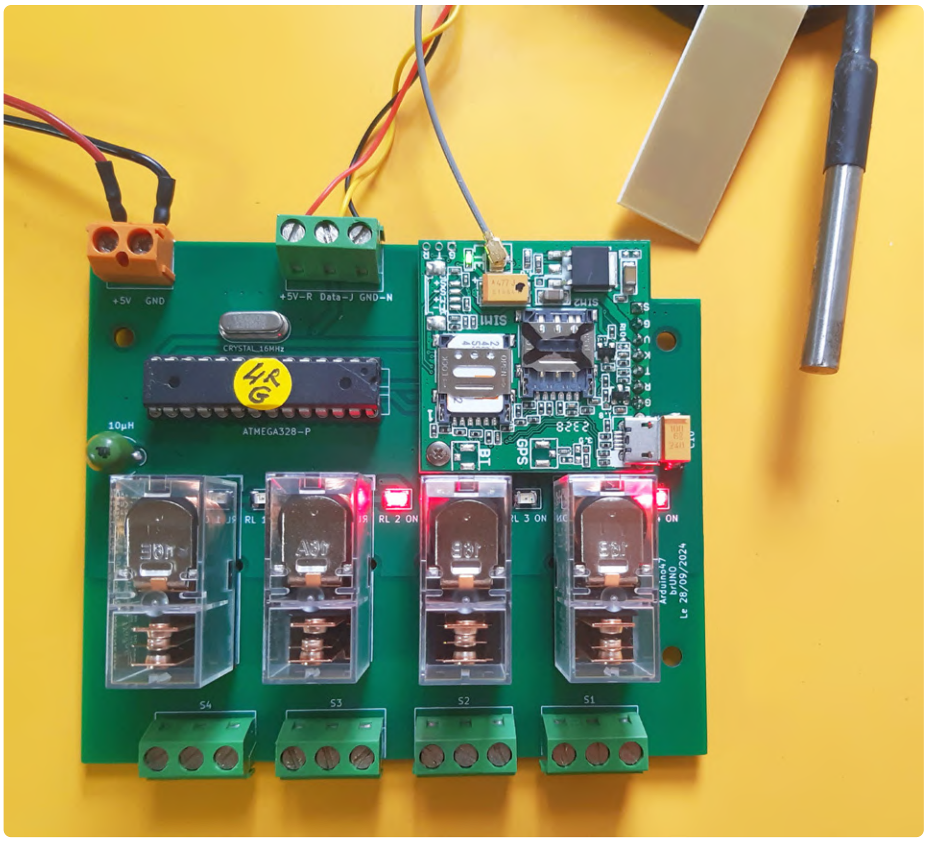


Figure 7: Latest assembled version, with the 4G antenna and DS18B20 sensor visible.

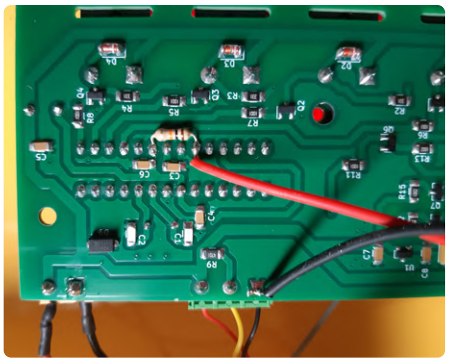


Figure 9: The buttons were added later and are directly connected to the pads.

controlled directly via push-buttons. This is very useful, for example, for testing the installation or manually switching relays if the GSM network is unavailable. The four buttons from a salvaged PCB are mounted on the front of the DIN-rail enclosure (**Figure 8**).

To reduce the number of microcontroller pins needed to read the push-buttons, I used a single analog input (ADC0/PC0). The method is based on a voltage divider with four resistors. Pressing each button grounds part of the divider, creating different voltages depending on the button pressed. The microcontroller uses ADC measurements to determine which button was pressed (corresponding to relay 1, 2, 3, or 4). Note: as this was a last-minute addition to the code made after the PCBs were manufactured, no pads are provided to connect the switches, so you'll need to install a 10-kΩ pull-up resistor and connect the BUS_BP and GND to the microcontroller beneath the PCB as per **Figure 9**.

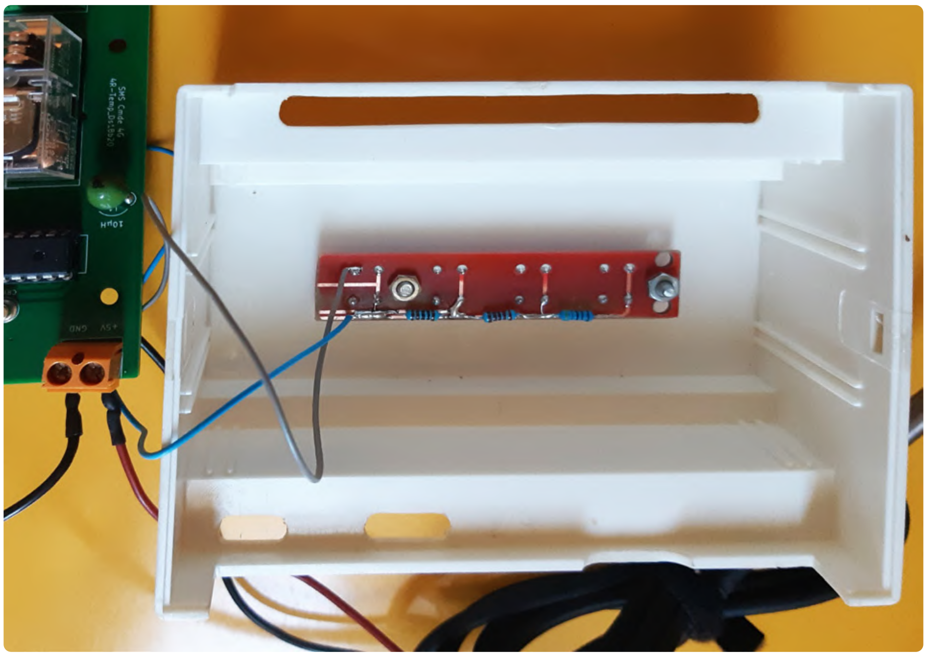


Figure 8: Front-mounted panel with four push-buttons.

Thanks to the new modem, it's now possible to include GSM signal strength (RSSI) in the status reply SMS. For this, a 5-LED signal strength indicator can be activated via the `LED_RSSI` option in the code. As for the switches above, you'll need to solder the LEDs, each with a 220 Ω resistor, directly to the corresponding microcontroller pads on the bottom of the PCB.

The design uses both through-hole and SMD components. The underside hosts the 3.3-V regulator, capacitors, control transistors, resistors, freewheeling diodes, and UART level shifters. The top side features SMD-mounted relay indicator LEDs and a 1000-µF capacitor, essential for stabilizing the 4G module's power supply and providing peak current when sending an SMS.

Configuring the BK-A7670E Module

By default, the BK-A7670E operates at 115,200 baud. To change it to 9600 baud, follow these steps. This configuration must be done before removing resistor R104, to allow independent power control without involving the Arduino. A SIM card is not required for this; the module should not be connected to your project or any external power source.

Connect the module to your PC using its USB port and wait for 20 s. Then, use terminal software such as SSCOM (which I use), RealTerm, or similar to connect at 115,200 baud. Send the command `AT+IPREX=9600`

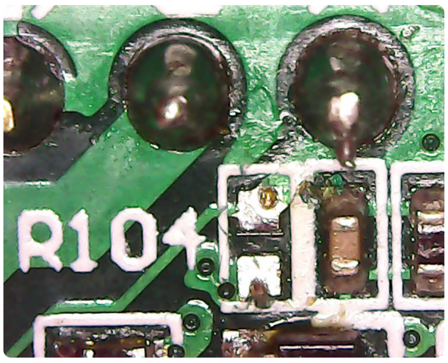


Figure 10: Remove resistor R104 on the BK-A7670E after configuring it.

and finally, disconnect the module. The UART speed is now fixed at 9600 baud. Once this is done, remove resistor R104 from the BK-A7670E (see **Figure 10**). This eliminates the default pull-down and allows the microcontroller to control power on/off; a 1 s pulse powers the module on, while a 2 s pulse turns it off.

Software

The code is too long to reproduce here, so I recommend checking it out on Elektor Labs. The latest version is available on [9], or you can explore its evolution through earlier links. In the `setup()` function, `init_Pins()` initializes the microcontroller's I/O pins. Then `power_On_Bk()` is called to power on the 4G module. The function `check_Appairage()` retrieves the values stored in EEPROM to restore the system's state after a power failure. The temperature sensor is initialized with `capteurDS18B.begin(ONE_WIRE_BUS)`.

After a delay of around 25 s, `start_Gsm_Bk()` is called to send AT commands to the BK-A7670E 4G module.

The program's first task is secure pairing via SMS. Once paired, it is possible to change the state of a relay using the SMS command `Relais X ON/OFF`, request the current temperature with the command `TEMP`, or obtain the system's global status using the command `ETAT`, which returns relay states, temperature, and signal strength (RSSI). By default, the program includes frost protection mode, as well as automatic modem restart in case of network loss. If your needs differ, you can modify the code accordingly. Among the optional features that can be enabled in the code are the local manual control via the push-buttons, and a display showing RSSI signal level using LEDs.

Given the BK-A7670E's affordable price (around €20 AliExpress) and the availability of inexpensive prepaid SIM cards, I hope this project inspires you to experiment and remotely control your own projects by SMS. By replacing the relay outputs with sensor inputs, this hardware could also serve well as a DIY alarm system, as I did at [11].

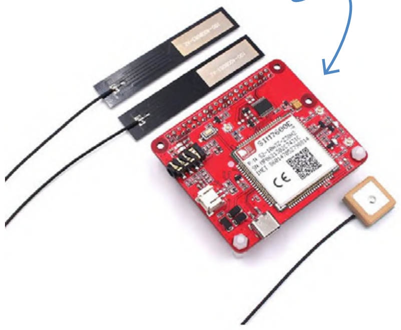
240585-01

Questions or Comments?

Do you have questions or comments about this article? Email the author at b.clerc31@laposte.net, or contact Elektor at editor@elektor.com.

About the Author

Bruno Clerc discovered electronics around age 12, thanks to his older brother. Curious about everything and thirsty for knowledge, he decided to study electronics at Bordeaux. He worked in tertiary low-voltage systems and then aeronautics and various other jobs. When microcontrollers arrived, not knowing programming, he concentrated on the maintenance of vintage hi-fi equipment. All this changed when, some years ago, his brother gave him an Arduino UNO. Bruno found a new passion and became "Arduino47." Today, he thanks the whole Arduino community that helped him advance in his learning.



WEB LINKS

[1] A variety of GSM remote control solutions, AliExpress: https://www.aliexpress.com/w/wholesale-gsm-remote-switch.html
[2] Read SMS commands with an Arduino Nano, F6KFA: http://f6kfa.fr/projet-de-telecommandes-via-le-reseau-gsm-par-envoi-de-sms/
[3] SIM800 module datasheet, SIMCom: https://soldered.com/productdata/2018/12/Soldered_SIM800-HW_datasheet.pdf
[4] Arduino Temperature Control Library, GitHub: https://github.com/milesburton/Arduino-Temperature-Control-Library
[5] First version on Elektor Labs: https://www.elektormagazine.com/labs/telecommande-par-sms-4-relais-et-temperature
[6] Update with new sensor and modem reboot, Elektor Labs: https://www.elektormagazine.com/labs/sms-remote-control-4r-v2
[7] Bruno Clerc, "Speed Controller for Fan or Ventilator", Elektor Circuit Special 2023: https://www.elektormagazine.com/220332-01
[8] A7670E documentation, SIMCom: https://nostris.ee/pdf/A7670%20Series%20Hardware%20Design_V1.00.pdf
[9] Adapter PCB for newer 4G modem, Elektor Labs: https://www.elektormagazine.com/labs/interface-pcb-pour-remplacer-le-sim800-par-un-bk-a7670e
[10] Latest version: everything fits on a single PCB, Elektor Labs: https://www.elektormagazine.com/labs/4g-is-served-la-4g-est-servie
[11] Alarm system based on a variation of the same hardware, Elektor Labs: https://www.elektormagazine.com/labs/tracker-4g

High-Speed Probe

High-Impedance Inputs for Signals up to 200 MHz

By Alfred Rosenkränzer (Germany)
Idea by Stefan Marenbach (Germany)

A few years ago, our author Alfred Rosenkränzer presented fast, active, and differential probes in Elektor. With a bandwidth of 1.9 GHz, they were indeed fast, but their input impedance of just 5 kΩ made them unsuitable for certain applications. The alternative presented here is not quite as fast but, with 500 kΩ single-ended and 1 MΩ differential, is sufficiently high-impedance for nearly all applications.

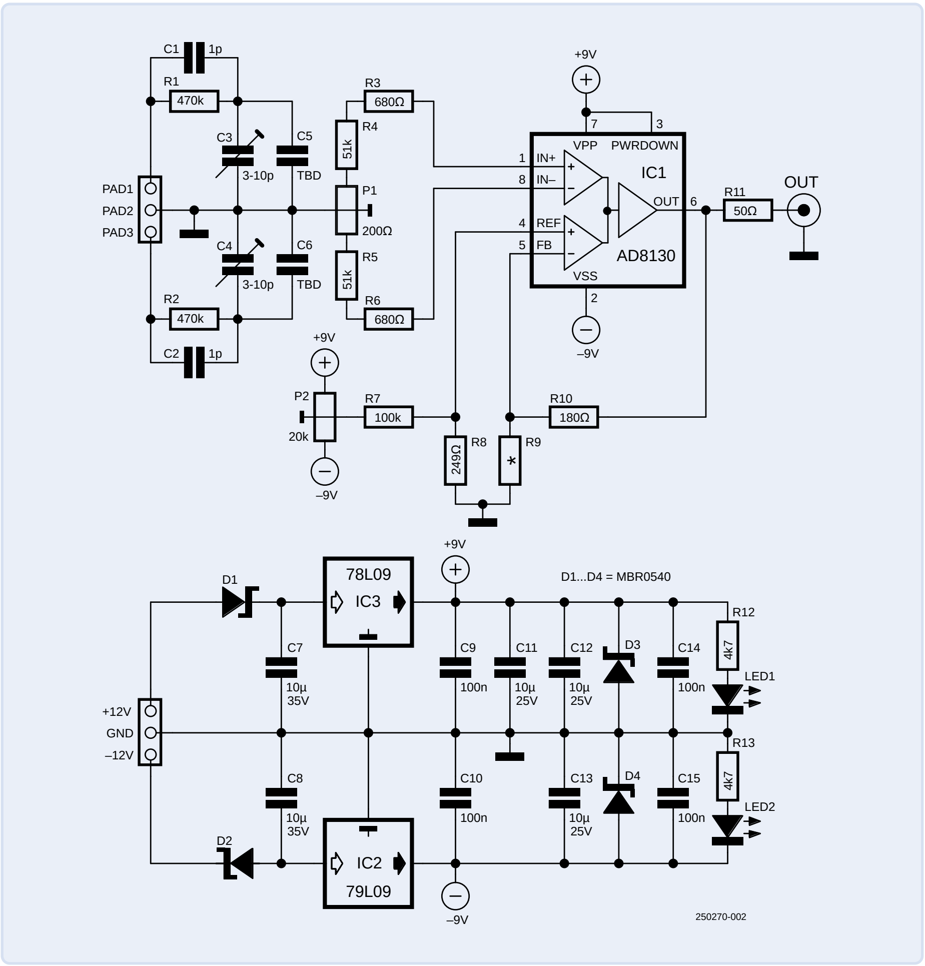


Figure 1: The central component of the circuit is a high-speed differential amplifier, type AD8130.

The predecessors of this probe [1][2] were specifically developed for measurements of fast differential signals such as LVDS, HDMI, USB, and so on. Their low input impedance leaves much to be desired, as it prevents low-interference measurements in many circuits. Thus, it is no surprise that Elektor reader Stefan Marenbach approached me with the idea of developing a higher impedance variant, even if that meant sacrificing some bandwidth. And here is the result: a still fast probe with a bandwidth of 200 MHz (-3 dB) and an input impedance of 500 kΩ or 1 MΩ, typical for use with oscilloscopes.

This new probe is a great match for affordable oscilloscopes with up to 200 MHz bandwidth, which are increasingly common in the maker scene. It offers an exceptionally low input capacitance of just 2 pF, meaning that RF signals measured with it experience minimal capacitive loading. The input voltage range of ±24 V is practical, and the self-noise is remarkably low. Building it yourself is worthwhile too, since ready-made solutions with similar performance are still very expensive.

Differential Amplifier

As shown in the schematic in **Figure 1**, the core of the circuit is the high-speed differential amplifier AD8130 from Analog Devices [3].

It is symmetrically powered with stable ±9 V using two small voltage regulator ICs (IC2 and IC3), as the supply voltage influences both the frequency response (which is flattest at ±9 V for this purpose) and the bandwidth (which reaches around 280 MHz at this voltage).

The differential input circuit on the left consists of two compensated voltage dividers with a 10:1 ratio. Potentiometer P1 compensates for the unavoidable resistor tolerances, as only with proper adjustment can good common-mode rejection (CMRR) be achieved. Trimmer capacitors C3 and C4 provide frequency compensation and ensure sufficient common-mode suppression at high frequencies. The adjustment of C3 and C4 largely follows the procedure used for regular passive probes and will be described later. Fixed capacitors may be used in place of — or in addition to — C3 and C4 in the form of C5 and C6, but this is generally unnecessary and they were therefore left unpopulated by me.

C1 and C2 form a defined input capacitance (parallel to R1 and R2). Their capacitance adds to that of the solder pads at the signal inputs.

If you're wondering what R3/R4 and R5/R6 are for: they are used to fine-tune the exact 10:1 ratio of the input voltage dividers. The formula is: (R1 + R3 + R4 + 1/2 × P1) / (R3 + R4 + 1/2 × P1) ≈ 0.1 ± 1%.

Potentiometer P2 adjusts the amplifier offset. The gain of the AD8130 is set by resistors R9 and R10. Since R9 is not populated, the gain is 1. The value of R10 influences the frequency response: at 180 Ω, the response is maximally flat. Higher values increase gain at high frequencies and vice versa. Since the AD8130 maintains a constant gain–bandwidth product, the bandwidth decreases as gain increases.

The maximum input differential voltage for the AD8130 is ±2.4 V, according to the datasheet [3]. Higher voltages will cause clipping. The maximum output current is 40 mA into 50 Ω.

Figure 2 shows the PCB layout. The associated design files are available at [4]. You can see the tight component placement, especially on the left side. The board still fits into the same enclosure used for earlier probes. A 50-Ω coaxial cable is soldered directly to the *Out* pads. For high-speed signals, the oscilloscope input should be terminated with 50 Ω. The attenuation factor of the probe is 20:1 with 50 Ω termination — and 10:1 with a high-impedance input.

Take great care when soldering the trimmer capacitors and potentiometers. Their leads should be pre-tinned before assembly.

Figure 3 shows the fully assembled PCB inside the opened housing.

Calibration

Start by shorting both inputs to ground and connect the probe output to an oscilloscope. First, use P2 to set the offset to exactly 0 V.

In the second step, both inputs are tied together and connected to ground through a symmetrical, DC-free square wave signal from a function generator. Adjust P1 until a flat line is displayed. **Figure 4** shows the output signal with incorrect adjustment. The spikes can be ignored — the important part is that no square wave should be visible, indicating symmetrical input adjustment.

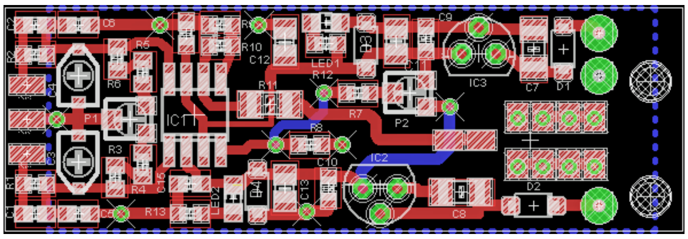


Figure 2: In the PCB layout, the components are especially densely packed on the left side.

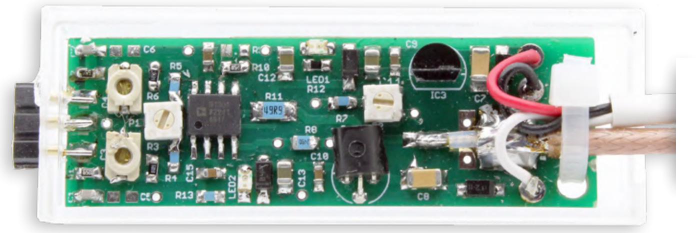


Figure 3: The fully assembled PCB in the opened enclosure.

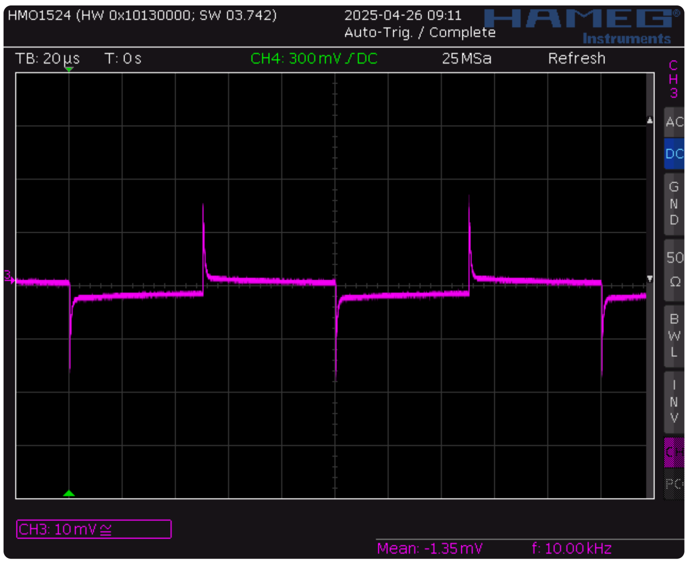

HMO1524 (HW 0x10130000; SW 03.742)
2025-04-26 09:11
Auto-Trig. / Complete
TB: 20 µs T: 0 s CH4: 300 mV ⁄ DC 25 MSa Refresh
CH3: 10 mV ≅
Mean: -1.35 mV f: 10.00 kHz

Figure 4: Symmetry adjustment with P1 has not yet been performed here.

Now for the frequency compensation: connect only one input to the square wave source while grounding the other. Adjust the trimmer capacitor for the active input to achieve the cleanest square wave signal — with no overshoot or undershoot — just as with a passive probe. **Figure 5** shows the signal before (top) and after (bottom) proper compensation. Repeat the same procedure for the second input.

Finally, as in step two, connect both inputs to a symmetrical, DC-free 1 MHz sine wave from a function generator. Slightly adjust C3 and C4, one at a time, until the visible signal's amplitude reaches a minimum. If needed, revisit the frequency compensation tuning afterward.

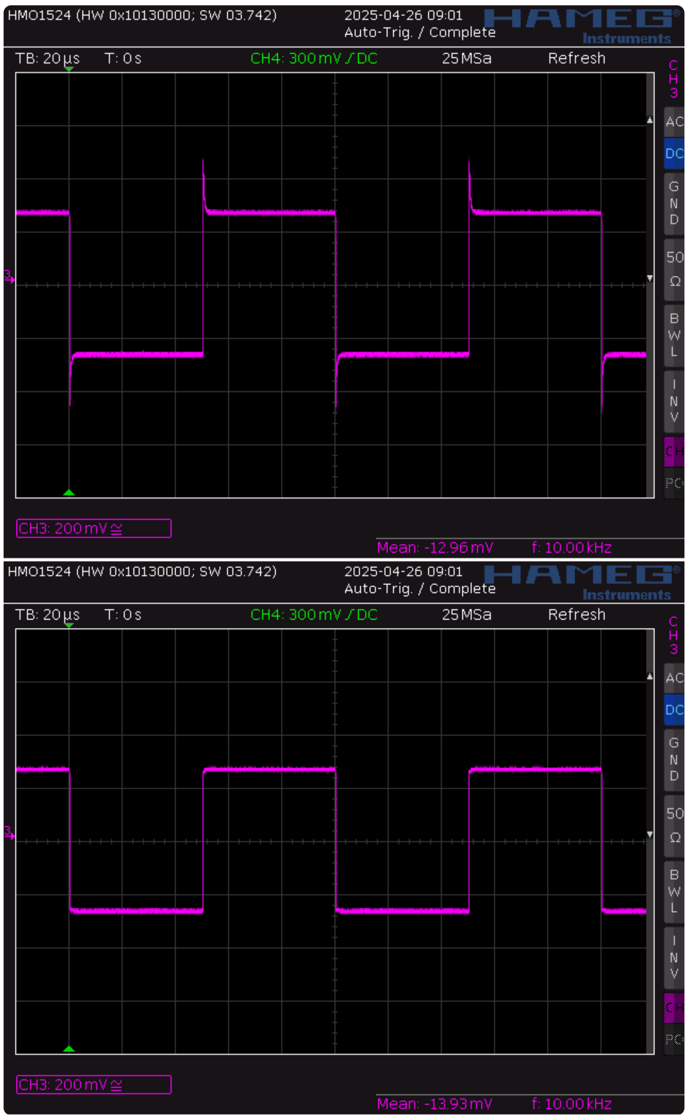


Figure 5: The signals before (top) and after successful frequency compensation (bottom).

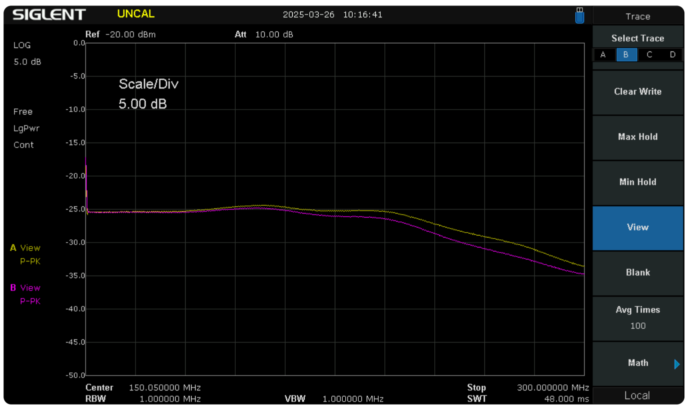


Figure 6: The frequency responses of both channels are stable up to over 200 MHz.

Additional Notes

At a supply voltage of ±9 V, the common-mode input range of IC1 is approximately ±7 V. Thanks to the 10:1 input voltage divider, this results in a usable common-mode range of about ±70 V, which should suffice for most applications. Be sure that input resistors R1 and R2, and especially capacitors C1 and C2, have adequate voltage ratings.

Figure 6 shows the frequency responses of both channels (single-ended), captured sequentially using a spectrum analyzer with tracking generator.

Translated by Jörg Starkmuth — 250270-01

Component List

Resistors

(Unless otherwise specified = SMD 0603)

R1, R2 = 470 kΩ
R3, R6 = 680 Ω
R4, R5 = 51 kΩ
R7 = 100 kΩ
R8 = 270 Ω
R9 = not populated*
R10 = 180 Ω
R11 = 50 Ω, SMD 1206
R12, R13 = 4.7 kΩ
P1 = 200 Ω, trimmer potentiometer, e.g., DigiKey 987-1573-1-ND
P2 = 20 kΩ, trimmer potentiometer, e.g,. DigiKey 987-1815-1-ND

Capacitors

(Unless otherwise specified = SMD 0603)

C1, C2 = 1 pF, 100 V
C5, C6 = not populated*
C7, C8 = 10 µF / 35 V, SMD 1206
C9, C10...C15 = 100 nF
C11...C13 = 10 µF / 25 V, SMD 0805
C3, C4 = 3...10 pF, trimmer capacitor, e.g., DigiKey 2447-SGC3S100CT-ND

Semiconductors

D1...D4 = MBR0540, Schottky
LED1, LED2 = LED, SMD 0805
IC1 = AD8130ARZ, SO8
IC2 = 78L05, TO92
IC3 = 79L05, TO92

Miscellaneous

Enclosure: Strapubox USB1, 56 × 20 × 12 mm, Conrad 531276-62
50 Ω coaxial cable with BNC connector on one end
3-pin female header, 0.1″ pitch
PCB

* see text

About the Author

Alfred Rosenkränzer worked for many years as a development engineer, initially in the field of professional television technology. Since the late 1990s, he has been developing digital high-speed and analog circuits for IC testers. Audio is his private passion.

Questions or Comments?

Do you have questions or comments about this article? Email the author at alfred_rosenkraenzer@gmx.de, or contact Elektor at editor@elektor.com.

Related Products

- **FNIRSI DPOX180H (2-in-1) 2-ch Oscilloscope (180 MHz) & Signal Generator**
  www.elektor.com/20640
- **OWON XSA815-TG Spectrum Analyzer (9 kHz – 1.5 GHz)**
  www.elektor.com/20484

WEB LINKS

[1] Alfred Rosenkränzer, "2-GHz Active Differential Probe," Elektor 7-8/2015: https://www.elektormagazine.com/magazine/elektor-201507/27980
[2] Alfred Rosenkränzer, "Active Differential Probe (V2)," Elektor 3-4/2017: https://www.elektormagazine.com/magazine/elektor-201703/40232
[3] Datasheet AD8130, Analog Devices: https://tinyurl.com/4nfcpwuw
[4] Download layout files from Elektor Labs: https://www.elektormagazine.com/labs/differentieller-high-speed-tastkopf-mit-ad8130

2 ACTIVATORS, 0 HALOGENS

almit
Technology for the future

DB-ZERO

... the new halogen-free counterpart to DB1-RMA

You are familiar with our innovative DB1-RMA solder wire, which has opened up a new dimension in wetting with its dual boost.

With our new DB-ZERO, you can now benefit from the advantages of DB1-RMA in a halogen-free solder wire without having to make any compromises.

- *Fast wetting*
- *High SIR value*
- *Hardly any spatter*
- *No tip formation*

Achieve maximum efficiency with minimum environmental impact.

DUAL BOOST SERIE

FAST ACTIVATOR | *PERSISTENT ACTIVATOR*

productronica
VISIT US 18. – 21. Nov. 2025
STAND / BOOTH **A4.454**

Almit GmbH Unterer Hammer 3 64720 Michelstadt +49 (0) 6061 96925 0 www.almit.de info@almit.de

Figure 1: Suddenly without power... (Source: Adobe Stock / Antonioguillem)

From Life's Experience

Kafka

By Ilse Joostens (Belgium)

In the January issue of this year [1], I wrote about plug-in solar panels, which, unlike in neighboring countries, were strictly forbidden in Belgium for a long time. As of April 17, they are finally officially allowed. But don't cheer just yet — this is Belgium, the land of Magritte. And the new legal framework for plug-in solar panels and plug-in home batteries isn't free from surrealism either.

With a bit of imagination, I can picture how the meeting must've gone at the umbrella organization for electricity and gas grid operators. Something like: "We're getting more annoying questions from people who think Belgium is lagging behind our neighbors, where plug-in solar panels have been allowed for years." "If everyone can just use their own energy that easily, it's going to cost us a lot of money." "I propose we allow it — unfortunately, we have no choice — but we make it hard enough so people give up." "Maybe we should change the regulations a few times along the way and come up with extra technical requirements." "We could also involve external actors, like property managers or city planning departments." "If they can block solar panels because of historic building views or the uniformity of apartment buildings, then it's not on us." "Great, that's settled — onto the next agenda item."

Hideous Heritage

As of March 17, the sale of plug-in solar panels and plug-in home batteries was allowed, and exactly a month later, you could actually plug them in [2][3]. As long as the total output stayed under 800 W, no procedure was required to start using them. Then suddenly, at the end of March, a registration requirement dropped out of the sky for anyone still using an analog meter [4]. Those meters might just start running backward and give users a tiny advantage. Once you register, within three months you'll be "blessed" with a smart meter — supposedly for the safety of grid operator staff (**Figure 1**). As if one little solar panel could single-handedly power an entire neighborhood. And besides, it has to shut down if grid power is lost anyway.

I'm not a big fan of the smart meter, and I think the capacity tariff is nonsense. Flip on a few too many devices at once for just 15 minutes, and you're hit with extra charges for the whole month. That even applies on a sunny weekend afternoon when the grid can easily handle the surplus. So, better to be frugal. These smart meters can also remotely shut off your power without a technician ever visiting your home. Grid operators promise to use this feature responsibly, but it makes energy users more vulnerable to administrative mistakes — or even cyberattacks [5][6]. A friend of mine runs a B&B in Spain, and her jealous ex canceled her electricity contract without her knowing. A technician came early in the morning and cut the power at the meter, which was mounted on the outside of the house. Suddenly, my friend and her guests were left without power (**Figure 2**), and it took several days to fix, resulting in serious losses — from spoiled food to unhappy customers. If the technician had just rung the doorbell, it could have been avoided. And with smart meters, these kinds of stunts are even easier to pull off.

At the end of March, the Union of Property Managers sent a letter to all its members warning them about the arrival of plug-in solar panels. In most Belgian apartment buildings, using such panels is ruled out by co-ownership and internal building regulations, which dictate things like the building's appearance. Ironically, these panels are popular with apartment dwellers in other countries. Several municipalities are also considering stricter zoning rules to block plug-in panels, and mayors — especially on the coast — are wary of apartment facades cluttered with solar panels [7]. Never mind that Belgium's coastline is already considered the ugliest in the world (**Figure 3**). One Reddit user even called

Figure 2: Smart meter. (Source: Adobe Stock / Joeri)

Figure 3: The ugliest coastline in the world. (Source: Adobe Stock / Thomas)

Belgium "a concrete cancer on the surface of planet Earth." Hannes Coudenys wrote two books about Belgium's hideous architecture, titled *Ugly Belgian Houses* [8]. I take all this with a grain of salt, of course. But if we're supposed to reduce every last gram of CO_2 to avoid climate disaster, then a few more solar panels shouldn't hurt. Besides, the Belgian tax authority already considers coastal apartments uninhabitable in the long term because climate change will almost certainly raise sea levels by more than a meter [9].

To EnFluRi or Not to EnFluRi

Even if you want to use a plug-in home battery, don't expect it to be simple, because they're not exactly "plug-in" in Belgium. You're required to install an EnFluRi sensor in your breaker box. I had to look that up — it stands for "EnergieFlussRichtung" (Energy Flow Direction), essentially an energy flow meter that communicates with the battery via Modbus or wirelessly. If you need a technician to install it, the costs and payback period can become quite steep. In the Netherlands, you can just use the P1 port of your smart meter combined with a wireless dongle — anyone can do it. But in Belgium, the P1 port is only allowed for monitoring and home automation, not for controlling devices that impact the grid or the direction of energy flow. A firmware update to the smart meters could make this technically possible in Belgium, too. The relevant minister is currently examining whether the use of the P1 port could be permitted after all. Kafka at its finest.

250423-01

WEB LINKS

[1] Ilse Joostens, "From Life's Experience — Micromanagement", Elektor 1-2/2025 Bonus Edition: https://www.elektormagazine.com/power-energy
[2] Plug-in solar panels and plug-in batteries in Flanders, Vlaanderen.be [Dutch]: https://tinyurl.com/plug-in-solar-panels
[3] Plug & Play photovoltaic panels: a new opportunity?, Wallonie Énergie [French]: https://tinyurl.com/plug-play-pv
[4] Important requirement for those purchasing plug-and-play solar panels, redactie24 [Dutch]: https://tinyurl.com/requirement-plug-play-pv
[5] OSU research shows how hackers can target smart meters to destabilize electricity grid, Oregon State University: https://tinyurl.com/osu-smart-meters
[6] How (and why) cyber specialists hacked a North American utility's smart meter, Cyberscoop: https://cyberscoop.com/mandiant-utility-hack-smart-meter-red-team/
[7] Municipality of Koksijde launches joke of the year, P-magazine [Dutch]: https://p-magazine.com/nl/articles/gemeente-koksijde-lanceert-grap-van-het-jaar
[8] Ugly Belgian Houses: https://uglybelgianhouses.tumblr.com/
[9] Remarkable ruling by the tax authority: "Depreciation of renovation costs for coastal apartments not allowed because flat will become uninhabitable due to rising sea levels", Business AM [Dutch]: https://tinyurl.com/ruling-by-tax-authority
[10] Test-Aankoop still sees many obstacles for plug-and-play devices: "Why is Belgium making it so difficult?", test-aankoop.be [Dutch]: https://tinyurl.com/test-aankoop-plug-play

KrakenSDR

By Robert Lacoste (France)

KrakenSDR includes no fewer than five RF receivers, which can be used independently — for example, tuned to different frequencies. Additionally, you can employ it for pinpointing a specific radio transmitter using a technique known as angle-of-arrival (AoA) detection. By measuring the phase difference between the respective receivers, you can determine the angle between the antenna array and the transmitter. In addition, this platform can be used for various other exciting experiments, such as passive radar.

Please note: This review was written for our website elektormagazine.com in 2023. The review is still a good starting point for experiments with the device, but things may have been updated since.

A Phase-Coherent SDR Receiver

There are plenty of low-cost SDR receivers on the market, but the KrakenSDR [1] is a more specific device: It is a multichannel phase-coherent receiver. What does this mean? Multichannel means that it includes not one but several RF receivers, specifically, five receivers. These receivers can be used independently, for example, tuned to different frequencies. However, the real added value is that these receivers are all synchronized by the same clock source, and the KrakenSDR includes circuits and firmware to compensate for any phase difference between the channels. That's the essence of phase-coherent. This ensures that the relative phase of the signals coming from the five inputs can be accurately assessed when the receivers are tuned to listen to the same transmitter.

Which Applications?

Alright, but what can you achieve with such a product that's not possible with a regular SDR receiver? First and foremost, you can employ it for pinpointing a specific radio transmitter using a technique known as angle-of-arrival (AoA) detection. The principle is straightforward: When a radio wave is received by multiple antennas, by measuring the phase difference between the respective receivers with a KrakenSDR, you can determine the angle between the antenna array and the transmitter. **Figure 1** illustrates this with a simple example using two antennas.

Angle measurement requires a minimum of two antennas. However, better performance can be achieved with more antennas, either in the form of a set of regularly spaced patch antennas or arranged in a circular pattern for 360° detection. A signal processing algorithm (the most well-known being MUSIQ) is then employed to deduce the most probable angle from the various signals, improving the signal-to-noise ratio. This is why KrakenSDR is equipped with five channels.

Of course, determining an angle alone is insufficient for locating a transmitter; you need at least two measurements from different locations to pinpoint its position. This can be accomplished by either installing a KrakenSDR in your vehicle and moving to obtain different angle measurements, which can be plotted on a map (KrakenSDR offers a convenient Android application for this, more details on that later), or by deploying several KrakenSDR units at distinct locations and sharing the results (KrakenSDR is also developing a cloud application to facilitate this).

In addition to AoA, this platform can be used for various other exciting experiments, such as passive radar. More on that later.

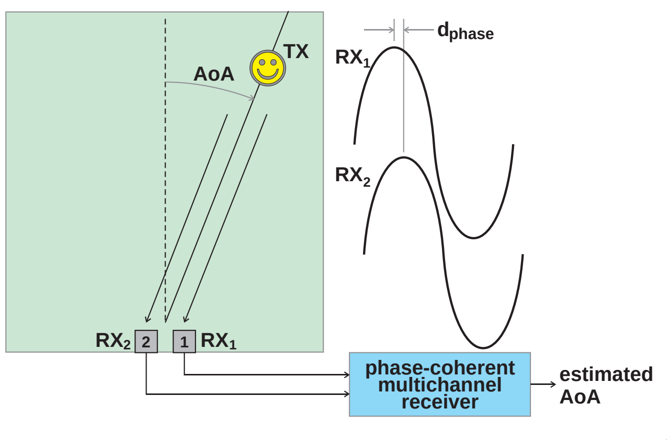


Figure 1: With KrakenSDR, you can determine the angle between the antenna array and the transmitter by measuring phase differences between antennas.

Specifications

Here are the key specifications of the KrakenSDR. Essentially, it comprises five standard RTL-SDR receivers, all synchronized to a single local oscillator. It also features automatic coherence synchronization hardware and a USB hub for connecting all five receivers to a single computer port.

Similar to all RTL-SDR receivers, each input can be tuned from 24 MHz to 1766 MHz (with the potential for a slightly higher frequency range with modified drivers). It boasts a reference clock based on a 1 ppm oscillator to ensure excellent phase stability. Additionally, the product includes a 4.5 V bias tee on each RF port, allowing for the powering of an external active antenna when necessary. While the RF performance details are not explicitly provided, they likely align closely with other top-tier RTL-SDR devices using the same chipset.

KrakenSDR features two USB-C connectors, one for data and one for power (it cannot be powered through the data port). It also offers five SMA female RF input ports. The product is enclosed in a sturdy aluminum case with a built-in heat sink and even a small fan, although no significant heat generation was detected during testing.

It's worth mentioning that KrakenSDR is an upgraded iteration of another product, Kerberos SDR, with the notable addition of a fifth channel and automatic calibration.

Unboxing and Essential Extras...

The KrakenSDR arrives in a straightforward yet efficient foam-protected cardboard box. Inside, you'll find a brief paper notice that essentially recommends consulting the latest documentation on *wiki.krakenrf.com* rather than relying on the printed documentation (**Figure 2**).

Now, here's a heads-up: You'll need to source a few essential extras yourself to make the most of your KrakenSDR, particularly if you intend to use it for AoA applications.

Figure 2: The KrakenSDR comes in a simple box.

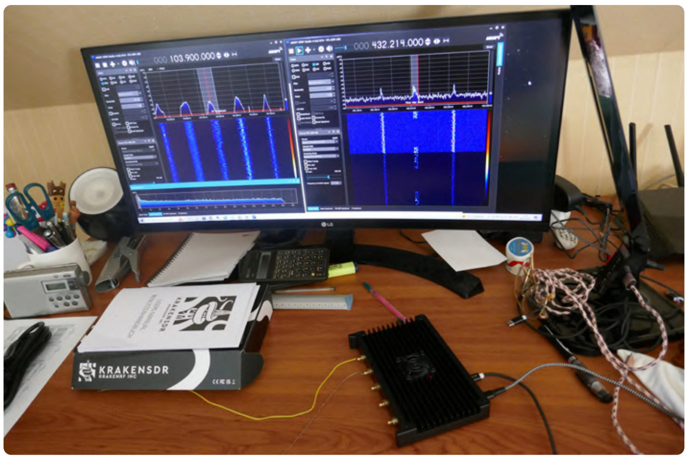


Figure 3: Simultaneously receiving an FM broadcast station at 105.5 MHz in WFM mode, as well as elusive transmissions in the 433 MHz ISM band.

Firstly, you'll require a USB power supply (at least 2.4 A/5 V) and two USB Type-C cables. Additionally, you'll need a computer to run the provided acquisition and signal processing software. This computer can be any system running Linux with a minimum of four cores at 2 GHz or higher. Several platforms have been confirmed to work, including Intel i5/i7 with Ubuntu, Raspberry Pi 4, Orange Pi 4/5, as well as Virtual Box Ubuntu on a Windows PC, although with some precautions. The simplest option appears to be using a Raspberry Pi 4, as ready-made SD images are available.

If you plan to utilize the companion client application, you'll also need an Android smartphone or tablet equipped with Wi-Fi, GPS, and a compass.

Finally, you'll need an antenna array. KrakenSDR can be purchased with the accompanying Krakentenna magnetic whip antenna set, but you can, of course, opt to construct the antennas yourself, as explained below.

A personal note to consider: While the choice of antennas depends on your specific project, it's worth mentioning that including cables and a power supply with the KrakenSDR kit could be a convenient option. This addition wouldn't substantially increase the price and would simplify the purchasing process for the buyer.

KrakenSDR as a Basic SDR

Why not begin by using the KrakenSDR as five independent RTL-SDRs communicating through a single USB connection? Since each RTL-SDR appears as a distinct serial number (0 to 4) on the PC, you can utilize any RTL-SDR compatible software to use them as regular SDR receivers.

To perform this test, I initially downloaded and installed one such application, SDR# [2], on my Windows PC, along with the RTL-SDR drivers at [3]. I then carefully followed the instructions at [4]. A few minutes later, I was ready to go. I launched SDR#, and it automatically started on receiver 0. Then, I launched another instance of SDR#, and it activated itself with receiver 1, and so on. I simply connected a few inches of copper wire to the RF ports as makeshift antennas, and I could simultaneously receive an FM broadcast station at 105.5 MHz in WFM mode, as well as elusive transmissions in the 433 MHz ISM band, as shown in **Figure 3**. It's a neat feature, even if not frequently used, isn't it?

Creating an Antenna Array

Now it was time to explore the most anticipated feature: AoA measurement. But, to accomplish this, an antenna array was necessary. KrakenSDR supports two antenna patterns: linear and circular. The former provides superior angular accuracy, while the latter allows unambiguous detection across a 360° span. I opted for the circular option. KrakenSDR offers a spreadsheet to streamline the calculations [5] (located at the bottom of the page). You only need to input the desired RF carrier frequency to generate several proposals with varying spacing multipliers, with 0.3 being a good starting point. For my test, I chose to operate at 868 MHz with a spacing multiplier of 0.3, resulting in an array radius of approximately 8.8 cm.

You can either purchase five antennas tuned to your target frequency and position them on a plane according to the calculated positions, or construct the array yourself, as shown in **Figure 4**, which is not overly complex. Given the relatively small size of the antenna array at 868 MHz, I used an old blank PCB laminate. I drilled five holes and created five pads on these holes using a small mill. Then, I soldered five vertical copper conductors with a length of a quarter wavelength (about 8.6 cm at 868 MHz) and attached five SMA-ended RG58 cables on the back. The key consideration is ensuring that the cable lengths are well-matched and that both copper sides of the PCB are thoroughly interconnected through multiple vias to ensure good RF performance.

Utilizing the Angle of Arrival Feature

On the software front, I initially attempted to use my PC within a Virtual Box environment with the provided Ubuntu VM image, but I encountered difficulties. Regrettably, the KrakenSDR failed to establish reliable communication with the PC.

While it should have been feasible, I didn't investigate the issue further and opted to use a Raspberry Pi 4 as my computing platform, which proved to be a much simpler alternative. I downloaded the SD image *krakensdr_pi4_060923.img.xz* [6] and imaged it onto a 64 GB class-10 SD card using Balena Etcher [7]. I inserted the SD

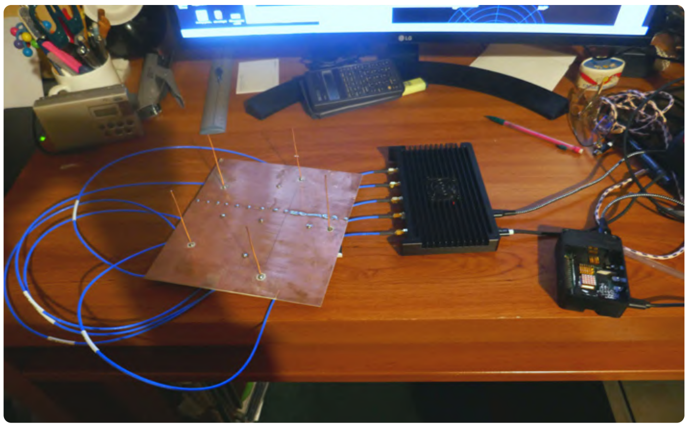
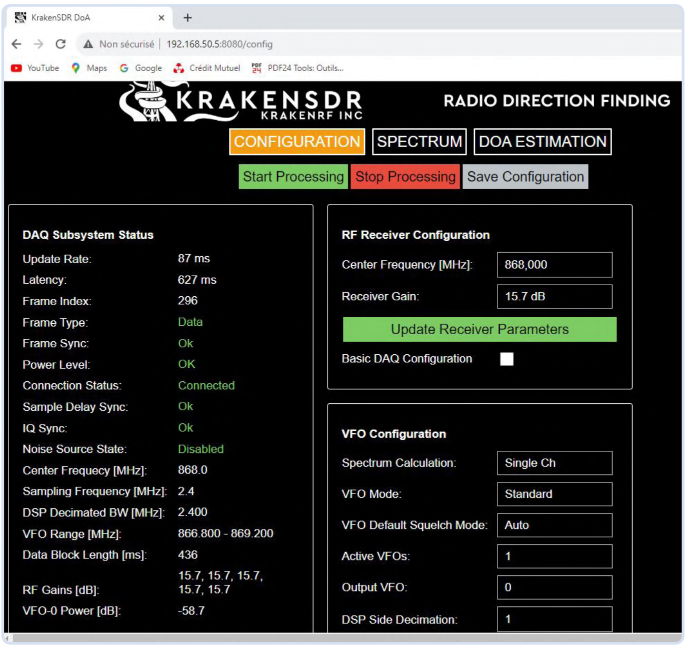


Figure 4: Antenna array built on a blank PCB laminate with five vertically soldered copper rods and RG-58 cables (SMA connectors) at the back.



DAQ Subsystem Status

Update Rate:	87 ms
Latency:	627 ms
Frame Index:	296
Frame Type:	Data
Frame Sync:	Ok
Power Level:	OK
Connection Status:	Connected
Sample Delay Sync:	Ok
IQ Sync:	Ok
Noise Source State:	Disabled
Center Frequecy [MHz]:	868.0
Sampling Frequency [MHz]:	2.4
DSP Decimated BW [MHz]:	2.400
VFO Range [MHz]:	866.800 - 869.200
Data Block Length [ms]:	436
RF Gains [dB]:	15.7, 15.7, 15.7, 15.7, 15.7
VFO-0 Power [dB]:	-58.7

RF Receiver Configuration

Center Frequency [MHz]:	868,000
Receiver Gain:	15.7 dB
Basic DAQ Configuration	☐

VFO Configuration

Spectrum Calculation:	Single Ch
VFO Mode:	Standard
VFO Default Squelch Mode:	Auto
Active VFOs:	1
Output VFO:	0
DSP Side Decimation:	1

Figure 5: A user-friendly web interface for configuring a direction-finding system.

card into the Raspberry Pi 4, powered it on, and patiently waited for about 2 or 3 minutes for the boot process to complete.

The Raspberry Pi 4 then established a Wi-Fi hotspot with the SSID "krakensdr" and "krakensdr" as the password. I connected my PC to this network and opened a web browser, navigating to *http://192.168.50.5:8080*, which is the default web page for the KrakenSDR web server on the Raspberry Pi. And there it was, a user-friendly web interface for configuring a direction-finding system, where I could set the frequency, gain, and other advanced settings as shown in **Figure 5**.

Just note that the frequency must be entered as "868,000." "868" or "868000" doesn't seem to work without any warning... Well, the software is probably still a work in progress, isn't it?

For this review, I conducted my experiments indoors. I used a small 868 MHz test transmitter with a diminutive antenna, configuring it at a very low power level (approximately -40 dBm). I moved it around the room. Admittedly, this wasn't the ideal setup for AoA measurement, as I likely had numerous signal reflections from the walls and furniture, but it was more convenient than venturing outside. After clicking *Start Processing*, I simply pressed the *Spectrum* button to ensure that the signal was well received and centered within the pass band (**Figure 6**).

Next, I clicked the *DOA Estimation* button and received a highly accurate angle-of-arrival estimate (**Figure 7**). **Figure 8** provides a closer look at the screen. The pattern roughly followed the transmitter's location, despite the presence of noticeable reflections in certain instances.

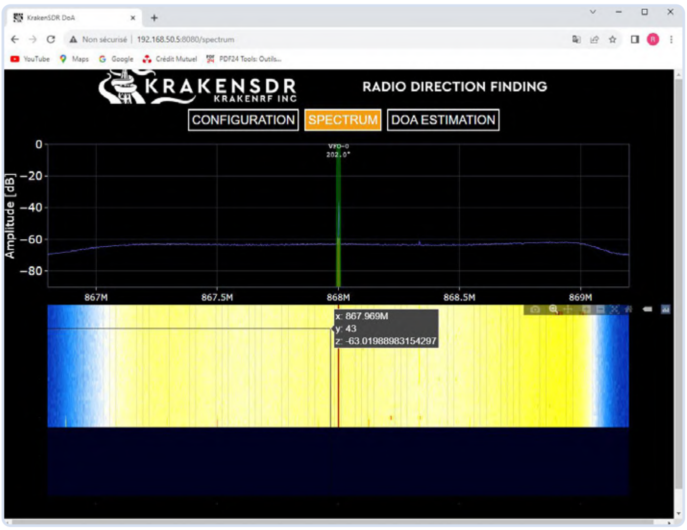


Figure 6: Press the Spectrum button to ensure that the signal is well received and centered within the pass band.

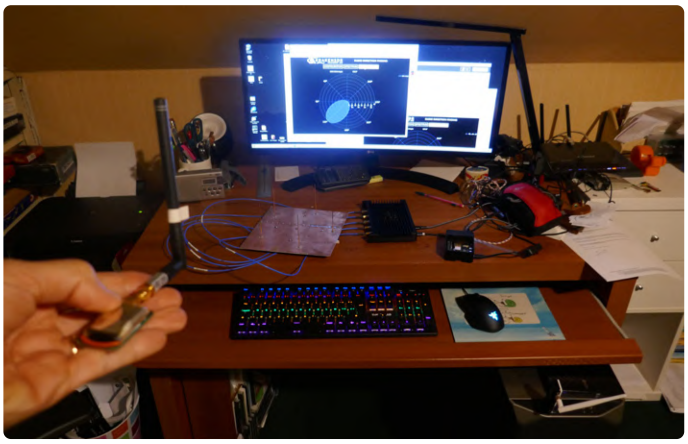


Figure 7: A highly accurate angle-of-arrival estimate.

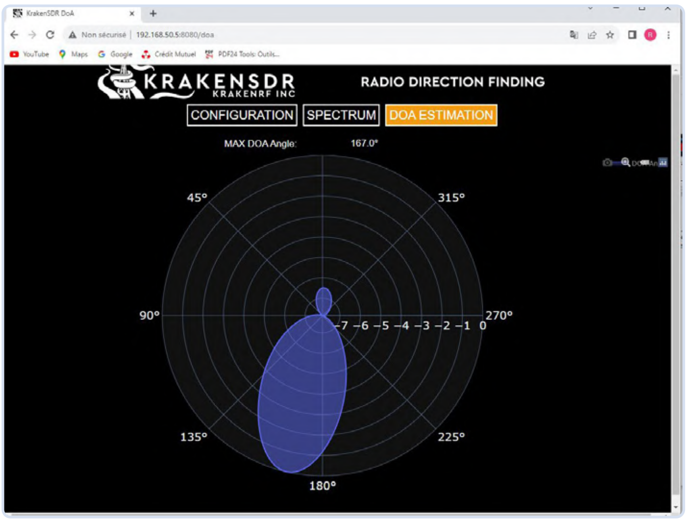


Figure 8: A closer look at the screen of the AoA estimate.

Android Application

As demonstrated, the web interface can be directly accessed from the Raspberry Pi web server for mobile direction finding. However, KrakenSDR also offers an Android application for mapping, data logging, and automatic transmitter location estimation. Unfortunately, being an Apple-oriented user, I couldn't test this part since there is no iOS version available at the moment.

In a nutshell, the app receives angle information from the Raspberry Pi and adjusts it based on the user's movement direction, determined through the Android phone's GPS and compass sensors. It then plots this data on a map. Accumulating enough measurements allows for the calculation of the transmitter's position, with the added potential for automatic turn-by-turn navigation to guide you to the transmitter. While untested, it certainly seems intriguing.

Finally, for distributed fixed and mobile sites, KrakenSDR is actively developing a cloud-based mapping solution [8] that will merge data from several measurement sites. This solution is currently in alpha testing.

Advanced Applications

The KrakenSDR platform is primarily marketed for AoA location applications, and its existing software is well-suited for this purpose. However, there are numerous other compelling experiments that can be conducted with a phase-coherent receiver like this. For such experiments, the product comes with a GNU Radio block to interface with the acquisition software, and both the acquisition and signal processing software are available as open source.

Passive Radar

One particularly interesting application is passive radar. In this scenario, the five phase-coherent channels are connected to five antennas placed at considerable distances from each other. Unlike traditional radar systems with a dedicated transmitter, passive radar uses signals from existing transmitters in the environment, such as TV stations and cellular base stations. It measures the phase difference between the signal received directly from the transmitter and the signal arriving via reflections from objects. This is a challenging task, but with five receivers, some exciting experiments become possible. KrakenSDR had previously provided experimental software for such applications, but it was removed from their web servers due to export regulations. To be continued...

A Peek Inside

One of the enjoyable phases when evaluating new hardware is grabbing a screwdriver and taking a look inside. With the KrakenSDR, this process is straightforward. I simply removed the four screws located beneath the unit's feet and disassembled the two parts, taking care with the fan wires (a bit of gentle force may be needed as heat transfer materials are sandwiched between the PCB and heat sink), as shown in **Figure 9**.

Next, I unscrewed the 20 small screws securing the PCB to the baseplate, revealing a well-designed PCB (**Figure 10**).

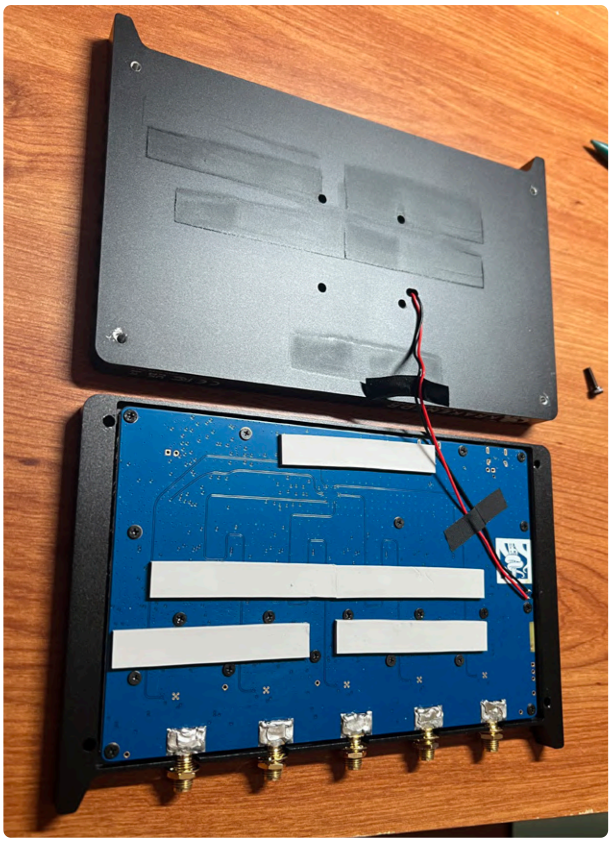


Figure 9: Taking a look inside at the KrakenSDR.

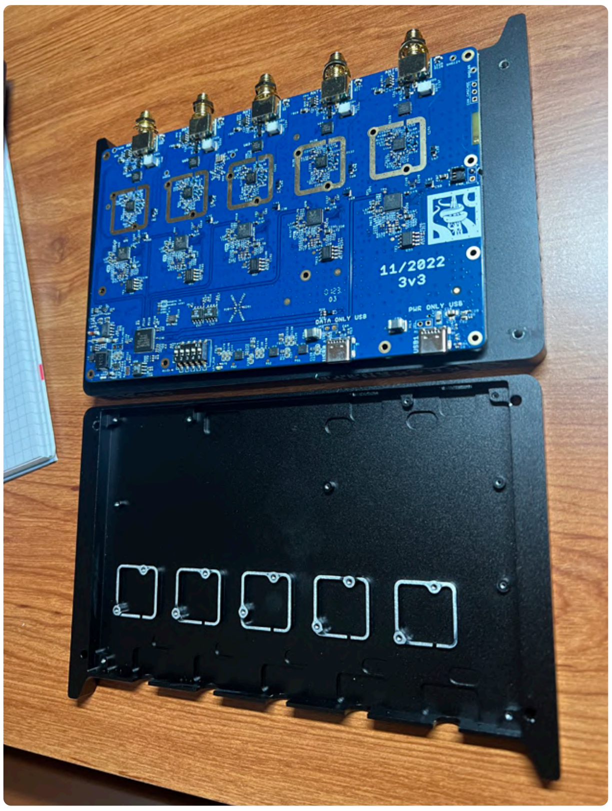


Figure 10: After more unscrewing, the well-designed circuit board is revealed.

It's worth noting that the lower part of the enclosure incorporates shielded cans for each of the five receivers, significantly reducing RF crosstalk between sections.

Taking a closer look at the PCB, the KrakenSDR features five identical RTL-SDR circuits, each composed of the classic R820T2 and RTL2832U chips. On the bottom, there's a USB hub chip, along with logic circuitry and power supplies. Furthermore, the RF ports include some ESD protections, which are not commonly found on standard SDR receivers (**Figure 11**).

Do You Need One?

Now, the question is: Do you need a KrakenSDR? If you're simply in the market for a single-channel SDR receiver, you'll likely find more cost-effective solutions with the same RF architecture and performance characteristics. However, if you're interested in experimenting with the direction of arrival location for signals spanning from HF to UHF bands, then the KrakenSDR is precisely what you need. Similarly, if you're an experimenter seeking to delve into passive radars or multichannel I/Q processing, this serves as an excellent starting point.

On the positive side, it's worth noting that the product exudes a remarkable level of professionalism and quality in its PCB and enclosure design. The extensive documentation available on the wiki and GitHub platforms is also a plus. In terms of frequency limitations, it's important to mention that the KrakenSDR can reasonably operate up to 1760 MHz, excluding the 2.4 GHz bands. However, this is a shared limitation among all SDR platforms based on RTL-SDR circuits.

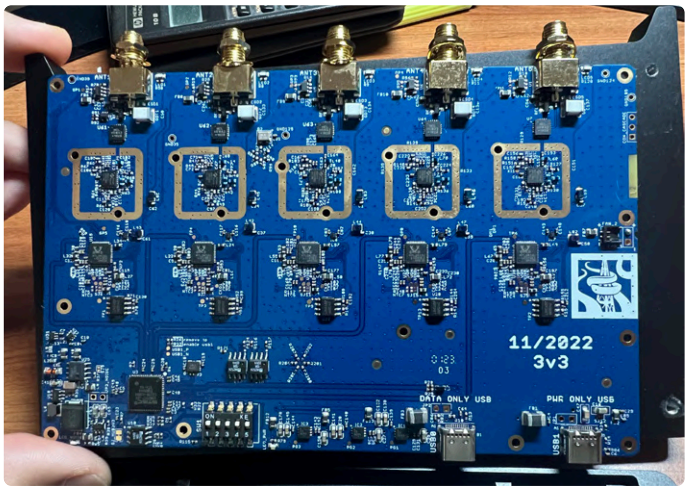


Figure 11: The KrakenSDR features five identical RTL-SDR circuits using R820T2 and RTL2832U chips. A USB hub, logic circuitry, and power supplies sit on the bottom. Notably, the RF ports include rare but welcome ESD protection.

On the downside, the software is still in its early stages, and occasional bugs may crop up, leading to the need for a few Raspberry Pi reboots. The good news, however, is that the software can be regularly updated.

Enjoy your experiments!

230658-01

Questions or Comments?

Do you have any questions or comments related to this article? Feel free to email the author at askrobert@lacoste.link or contact the Elektor editorial team at editor@elektor.com.

About the Author

Robert Lacoste lives in France, between Paris and Versailles. He has 40 years of experience with wireless systems, analog and digital designs, and signal processing. In 2003, he established his design-house company, ALCIOM, to share his passion for innovative mixed-signal designs. Robert is now an R&D consultant and trainer. He has written several articles for Elektor.

Related Products

- **KrakenSDR 5-channel SDR (based on RTL-SDR)**
  www.elektor.com/20070
- **Magnetic Telescopic Antennas for KrakenSDR (Set of 5)**
  www.elektor.com/20291

WEB LINKS

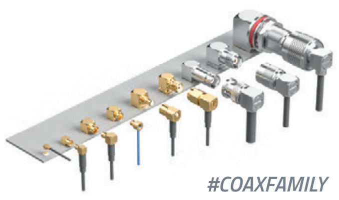
[1] KrakenSDR 5-ch SDR (based on RTL-SDR): https://elektor.com/products/krakensdr-5-ch-sdr-based-on-rtl-sdr
[2] SDR++: https://sdrpp.org
[3] Airspy: RTL-SDR drivers: https://airspy.com
[4] RTL-SDR instructions: https://rtl-sdr.com/rtl-sdr-quick-start-guide/
[5] GitHub Repository for Kraken SDR DoA: https://github.com/krakenrf/krakensdr_doa
[6] Image file for Pi 4 devices, GitHub: https://github.com/krakenrf/krakensdr_doa/releases/
[7] Balena Etcher: https://etcher.balena.io
[8] KrakenSDR Login: https://map.krakenrf.com/#/login

THE COAX FAMILY
HAS EXPANDED

WÜRTH ELEKTRONIK
MORE THAN YOU EXPECT

New coaxial products
Coaxial connectors and cable assemblies provide low loss paths between RF system, subassemblies or components. The coaxial product range includes frequency supporting up to 18 GHz and with various design.
www.we-online.com/coaxfamily

Highlights
- MIL-STD & IEC interfaces
- Characteristic impedance 50 Ω
- High quality data transmission
- Broad line of product portfolio with customize capability



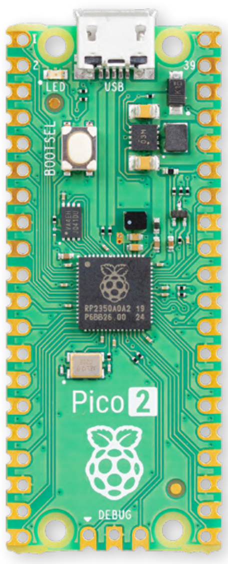
#COAXFAMILY

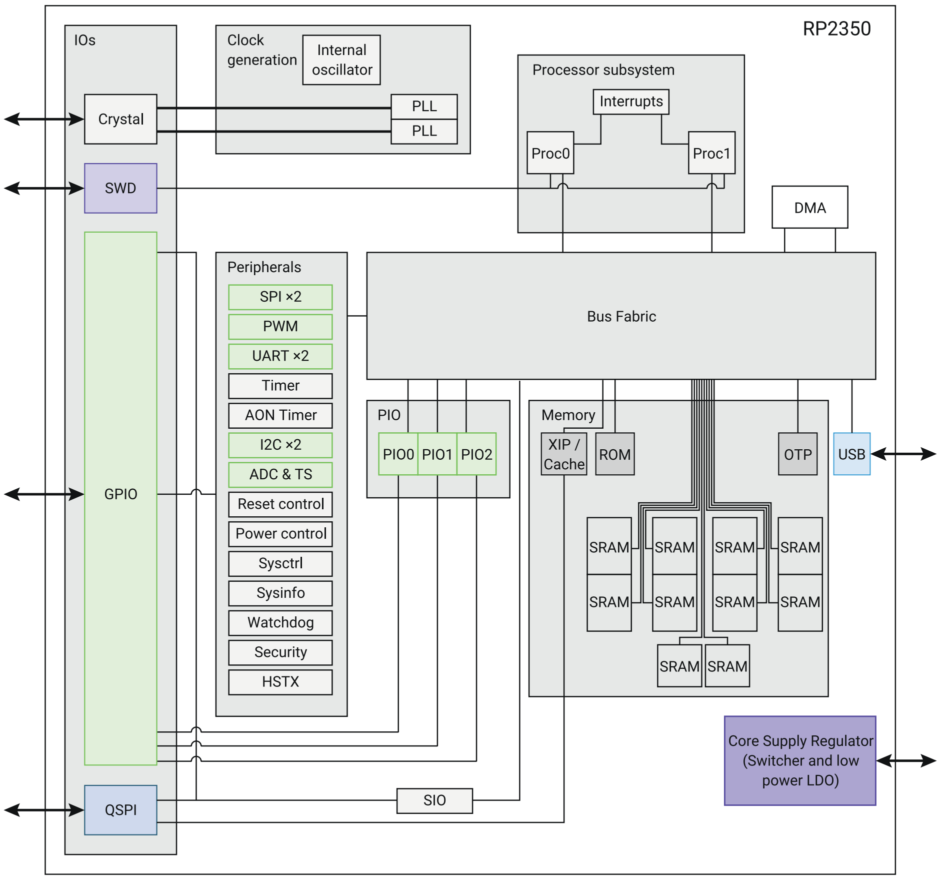
Performance Tests with the RP2350

Is an Upgrade from Raspberry Pi Pico 1 to Pico 2 Worthwhile?

By Prof. Dr. Martin Ossmann (Germany)

The RP2350 microcontroller, introduced with the Raspberry Pi Pico 2 as the successor to the Pico 1's RP2040, has been available for some time. It is of particular interest to determine the extent to which the new controller offers improved computational performance. This question is especially relevant given that the RP2350 can utilize both two ARM cores and two RISC-V cores. To provide clarity, a series of tests was conducted for this article using routines that frequently occur in practical applications.





Source: Raspberry Pi Ltd [6]

The following section outlines how to measure code execution time and what specific considerations apply when working with the Pico. Visual Studio Code (VSCode) with the Raspberry Pi Pico extension was used as the development environment. Further information is available in [1].

Time Measurement

A simple measurement method using an oscilloscope consists of setting a GPIO pin to high during execution:

```
gpio_put(debugPin,1);   // start output pulse
sleep_us(1);            // code to test
gpio_put(debugPin,0) ;
```

For testing purposes, we measure the routine `sleep_us(1)`, which should ideally take 1 µs to execute. The resulting oscilloscope trace is shown in **Figure 1**. A series of pulses with a duration of approximately 1 µs is visible. These are overlaid because the oscilloscope's persistence was set to infinite, allowing even single events to be displayed permanently. The execution time of the 1-µs routine fluctuates slightly. Additionally, two pulses with a duration of approximately 4 µs appear, which clearly cannot be attributed to normal runtime variations. There must be another cause.

After extended investigation, it was found that USB communication uses interrupts. If the interrupt service routine coincides with the 1-µs routine, the duration is correspondingly extended. Although this occurs infrequently, it can affect the actual measurement. If USB communication is disabled and the classic serial interface is used instead, this effect no longer occurs. Therefore, it is advisable to repeat time measurements multiple times to rule out such anomalies.

Those who do not have access to an oscilloscope can resort to a second method using the SysTick timer. The SysTick timer is a 24-bit counter that counts down with the system clock at 150 MHz. By reading its value before and after the code routine under investigation, the execution time can be determined from the difference. The corresponding code is as follows:

```
int time1=systick_hw->cvr;  // cvr: current value register
sleep_us(1);                  // code to test
int time2=systick_hw->cvr;
```

A value of 150 is expected (corresponding to 1 µs at 150 MHz); in practice, due to runtime fluctuations, values between 136 and 148 are observed. If an interrupt occurs, the value is significantly higher. With the 24-bit counter, durations of up to 0.1 seconds can be measured at 150 MHz. For longer durations, a custom 32-bit counter can be implemented using the PIO functionality [2]. This counter appears as a short PIO program as follows:

```
.program pioADCv01
.side_set 1 opt
.wrap_target
```

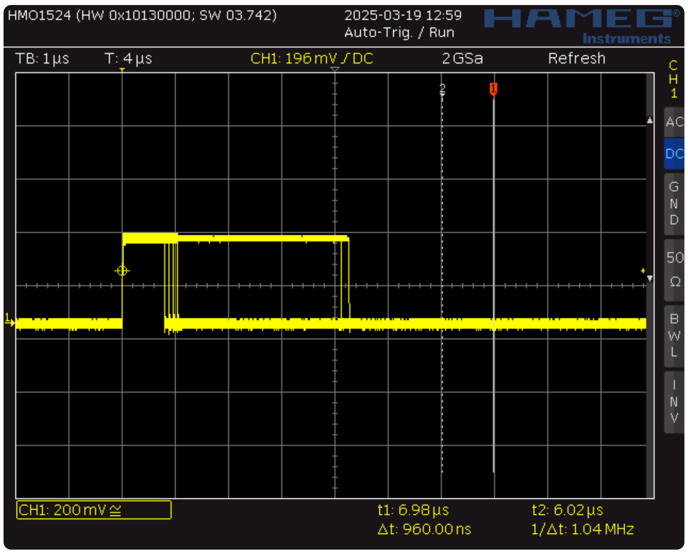


Figure 1: Oscillogram of the GPIO pin measurement.

```
lbl1:
   jmp X-- ,lbl2   side 0b0
lbl2:
   jmp X-- ,lbl1   side 0b1
   jmp lbl1
.wrap
```

The program mainly consists of two lines, each decrementing the X register and jumping to the next instruction. As a result, the counter is decremented by 1 on every clock cycle. Using the `side` annotations enables runtime monitoring by making the toggling at 75 MHz visible on a GPIO pin. The readout routine is somewhat more complex than with the SysTick timer:

```
int getX(){
  // Save X-counter value to input shift register isr
  pio_sm_exec_wait_blocking(pio, sm,
              pio_encode_mov(pio_isr, pio_x));
  // push pio_isr value in fifo
  pio_sm_exec_wait_blocking(pio, sm,
              pio_encode_push(0,0));
  // Get value from fifo
  return pio_sm_get_blocking (pio,sm) ;
}

void readXloop3(){
  while(1){
   lastVal=getX() ;
   sleep_us(1) ;
   val=getX() ;
   printf(" read lastVal-val=%10d\n",(lastVal-val) ) ;
   sleep_ms(500) ;
  }  }
```

This allows time intervals of up to 2^{32} / 150 MHz = 28 seconds to be measured.

For longer durations, it is often unnecessary to have single-cycle accuracy. If a resolution of 1 µs is sufficient, the 64-bit system timer, which operates at a 1 MHz clock rate, can be used. The corresponding code is as follows:

```
long long t1=time_us_64() ;
sleep_ms(1) ;
long long t2=time_us_64() ;
long long diff64=t2-t1 ;
```

A time interval of 1 ms was measured, resulting in a displayed value of 1000 (or 1001, which is merely a rounding fluctuation). For such long durations, the influence of USB interrupts and the cache mechanism (see below) is negligibly small. In general, it can therefore be stated that measuring short time intervals is more challenging than measuring long ones.

During performance measurements, it is sometimes observed that the first execution of a code segment takes significantly longer than subsequent executions. This is due to the fact that, during the initial run, the code must first be fetched from external flash memory into the cache. On subsequent runs, the code is already present in the cache and executes more quickly. This behavior is demonstrated by the following program listing, which executes the function `testFunction` five times and measures the execution time for each run:

```
volatile int tt = 0;

void testFunction(){
  for (int k = 0; k < 10; k++) { tt++; }
  for (int k = 0; k < 10; k++) { tt++; }
  }

int main(){
  printf("pico2XIPdemo2V01\n");
  for (int m=0 ; m < 5; m++){
    int t1 = time_us_32();
    testFunction();
    int t2 = time_us_32();
    printf(" us:%3d  \n", t2-t1);
    }
  }
```

During the first execution, the function requires 13 µs, while each subsequent execution takes less than 2 µs. This clearly illustrates the impact of the cache mechanism. Therefore, code must be executed multiple times in order to measure the true execution speed.

When comparing code performance on the Pico 2 (RP2350) with the Pico 1 (RP2040), it must also be taken into account that the Pico 2 can be clocked at 150 MHz, whereas the Pico 1 reaches its limit at 133 MHz. This alone gives the Pico 2 a speed advantage of approximately 15%.

As shown, reliable performance measurement is not straightforward. The identified issues and how to mitigate them were, of course, taken into account in the subsequent measurements.

Tests

The tests aimed to cover a cross-section of possible applications. The code can be downloaded from [3]. The following tests were performed:

simpleIntTest

The following code is executed, involving additions and one multiplication:

```
void doit(){
  vv=0 ;
  for(int k=0 ; k<1000 ; k++){
    for(int j=0 ; j<1000 ; j++){
      vv=vv+123 ;
      vv=vv+k+5*j ;
  } } }
```

intMatMul

The following routine for matrix × vector multiplication is executed:

```
nn=100;
for(int k=0 ; k<nn ; k++){
  int sum=0 ;
  for(int i=0 ; i<nn ; i++){
    sum=sum+aa[k][i]*xx[i] ; }
  yy[k]=sum ;
}
```

sortInt

A total of 1000 integers are sorted using the bubble sort algorithm [4].

```
void bubbleSort(int array[], int size) {
  for (int step = 0; step < size - 1; ++step) {
    for (int i = 0; i < size - step - 1; ++i) {
      if (array[i] > array[i + 1]) {
        int temp = array[i];
        array[i] = array[i + 1];
        array[i + 1] = temp;
      }
    }
  }}
```

floatFilter

The following routine for calculating the biquads of IIR filters [5] is executed 10 times:

```
float tprun(float x){
  float w,w11,w22,y ;
  for(int i=0; i< 6; i++){
    w11=w1[i] ; w22=w2[i] ;
    w=x-b1[i]*w11-b2[i]*w22 ;
    y=w+a1[i]*w11+a2[i]*w22 ;
```

Automation astounds.
With every new facet.

The SPS showcases it all and has been the home of automation since 1990.
It is the place where expertise unfolds, networks grow, and ideas are inspired.

Bringing together start-ups to global players and driving progress through diversity, expertise, and ingenuity.
For those who want to stay one step ahead.

Bringing Automation to Life

Discover more

Unfold the world of automation

34th international exhibition for industrial automation

sps

25 – 27.11.2025
NUREMBERG, GERMANY

mesago

Messe Frankfurt Group

Table 1: Measurement results of the various routines and processing cores.

	RP2350		RP2040
	ARM	RISC-V	
Int tests			
simpleIntTest	80	74	120
intMatMul	0.408	0.47	0.82
sortInt	28.8	25.4	51
Float tests			
floatFilter	0.013	0.19	0.36
floatFFT	1.4	18.6	40
floatMatMul	0.4	6	13
Double tests			
doubleFFT	5.8	76	88
doubleMatMul	3.7	23	24

All times in milliseconds.

```
    w2[i]=w11 ; w1[i]=w ; x=y ;
    }
  return a0*y ;
  }
```

floatFFT
A 1024-point FFT using float variables.

doubleFFT
A 1024-point FFT using double variables, i.e., double-precision floating-point numbers.

floatMatMul
Equivalent to *intMatMul*, but using float variables.

doubleMatMul
Equivalent to *intMatMul*, but using double variables.

Thus, tests were conducted involving integer values as well as float and double variables. With float variables, the advantage of the ARM cores in the RP2350 becomes evident, as they feature a Floating Point Unit (FPU) that significantly accelerates float computations compared to the RISC cores and especially to the RP2040, where such operations must be executed in software. In the case of double-precision calculations, the RP2350 also clearly leads, as it includes an acceleration unit for double operations. Although this does not achieve the speed of a dedicated double-precision FPU, the performance gain over the RP2040 is still substantial.

All measured times are listed in **Table 1**. In the integer tests, the ARM and RISC cores perform at approximately the same level and are nearly twice as fast as the RP2040 CPU. In the float tests, the ARM CPU of the RP2350 is more than ten times faster than the RISC CPU, clearly demonstrating the advantage of the FPU in the RP2350. The RISC CPU, in turn, is roughly twice as fast as the RP2040 CPU. In double-precision calculations, the ARM CPU is five times faster than the RISC CPU, which performs at approximately the same speed as the RP2040 CPU. This suggests that the RISC CPU of the RP2350 is somewhat slower in double operations compared to its relative performance advantage in integer or float computations.

Is the Upgrade Worthwhile?

The controller of the Pico 2 is twice as fast as the Pico 1's controller in integer computations. In float and double operations, the RP2350 CPU with ARM cores outperforms the RP2040 CPU by a factor of 5 to 25. The performance gain in integer processing alone makes the switch from Pico 1 to Pico 2 worthwhile.

Edited by Rolf Gerstendorf — 250238-01

Questions or Comments?

If you have technical questions or feedback regarding this article, please contact the author at ossmann@fh-aachen.de or the Elektor editorial team at editor@elektor.com.

About the Author

Martin Ossmann began reading *Elektor* — and tinkering, of course — at the age of twelve. After studying electrical engineering and working for several years as a development engineer, he became a professor in the Department of Electrical Engineering and Information Technology at FH Aachen. In addition to authoring scientific publications, he has been regularly publishing circuit designs and software projects in Elektor for over three decades, bringing extensive technical expertise to his contributions.

WEB LINKS

[1] Raspberry Pi Pico-series C/C++ SDK, Raspberry Pi: https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf
[2] Martin Ossmann, "RP2040 PIO in Practice," Elektor 3-4/2023: https://www.elektormagazine.com/220100-01
[3] Software download: https://www.elektormagazine.com/labs/elektor-articles-software-downloads
[4] Bubble sort, Wikipedia: https://en.wikipedia.org/wiki/Bubble_sort
[5] Digital biquad filter, Wikipedia: https://en.wikipedia.org/wiki/Digital_biquad_filter
[6] RP2350 Datasheet: https://datasheets.raspberrypi.com/rp2350/rp2350-datasheet.pdf

Always connected. Everywhere.

Reliable connectivity for mobile industrial applications – including railway use.

FORTEC INTEGRATED

www.fortec-integrated.de

Contact-Free E-Field Measurements (2)

A Laser Vibrometer for Assessing the Membrane's Vibrations

By Michael Monkenbusch (Germany)

In the first part of this article, we explored the possibility of making a measuring device for static electrical fields, with a contactless measurement of DC potentials of objects through a vibrating membrane, used as a sensor. In this second installment, we'll assess the amplitude of the membrane's vibration with the help of a laser-based makeshift vibrometer.

To get an independent measurement of the membrane amplitudes in contraptions used for the E-field detection described in the previous installment [1], I tried a makeshift interferometer to get an independent, rough value of the displacement magnitude. I was also curious how difficult it would have been to set up this little "light radar speedometer" for a moving object, as the vibrating membrane is. Thus, without reaching out to expensive optical and mechanical parts, a desktop household version was assembled and could be used with some success.

Operating Principle

The basic principle of the interferometer is illustrated in **Figure 1**. As the light source, a (red) laser diode — without its original lens — is used. The laser beam is then focused by a photo camera lens (here f = 50 mm) onto a spot on the object to be investigated, in our case the vibrating membrane.

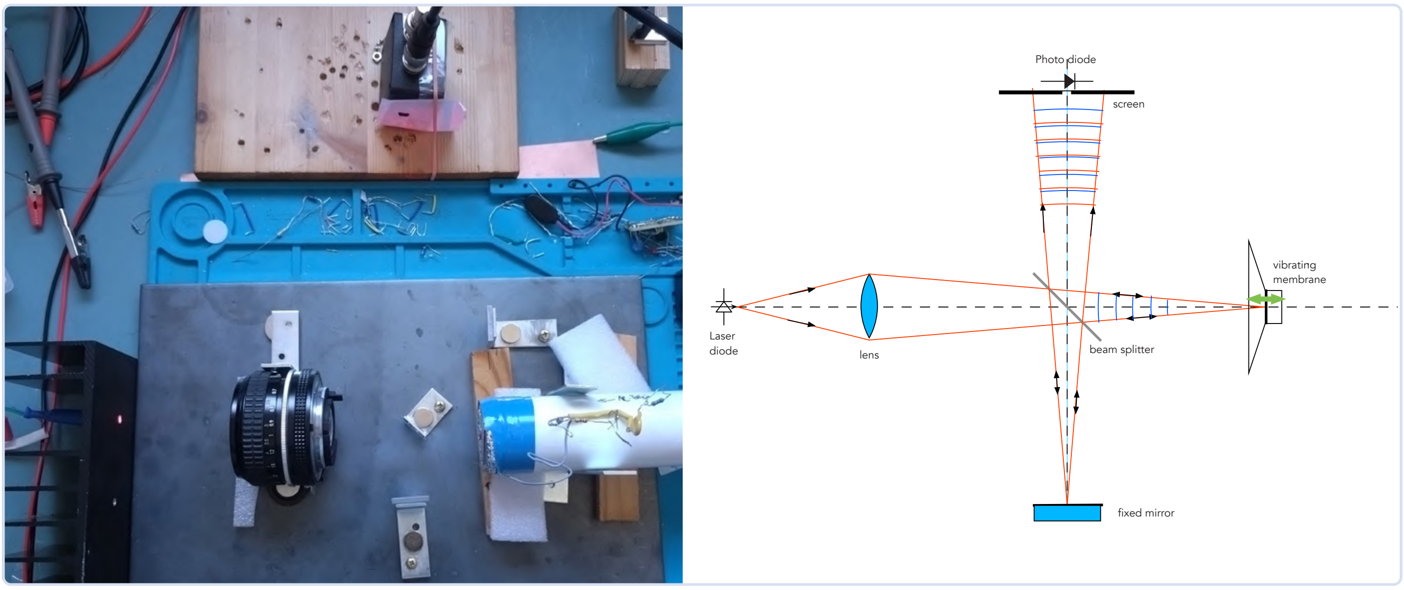


Figure 1: Photo and setup principle of the interferometer to measure the displacement/speed of a vibrating object.

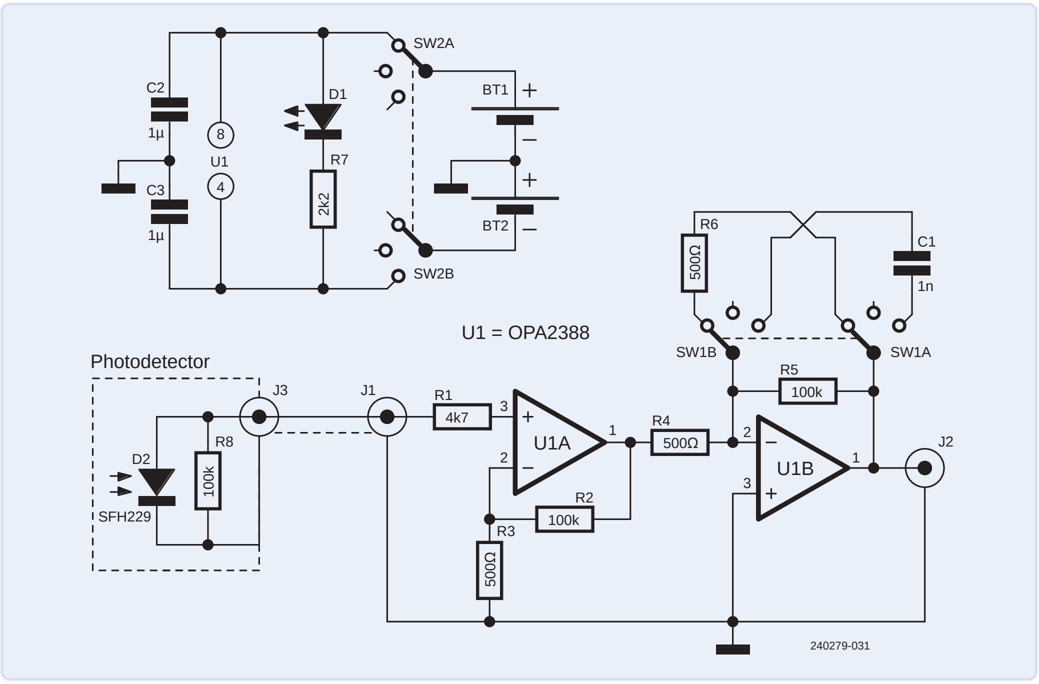


Figure 2: Schematic diagram showing the combination of the photo diode module and the optional 20× to 420× general purpose amplifier (both in separate aluminum enclosures), connected via a 50 cm coax cable. The output supplied at J2 is monitored by an oscilloscope.

Somewhere, in the light path, a beam splitter is inserted. This device reflects 50% of the laser beam onto a fixed mirror, which is also placed at focal distance. Now both the reflected light from this reference mirror (the 50% that can pass through the beam splitter and heads to the photodiode) and the light reflected by the vibrating membrane (the 50% that is reflected by the beam splitter) re-combine — i.e., interfere — at the photodiode target.

Depending on the path difference between the fixed mirror and vibrating membrane, the interference yields alternating maximum or minimum intensities each time the displacement advances by λ/2 ≃ 325 nm, where λ is the wavelength of the laser's light.

Practical Aspects

Let's spend here some words on my approach for quickly testing things that required some mechanical arrangement, such as mirrors, beam splitters and lenses.

On many occasions, it proved very handy to use a (magnetic) steel plate as a ground plane. Any items can then be very flexibly attached and positioned with the help of small, strong NdFeB magnets. Here, for the mirrors, etc., I used some standard angled iron pieces, as needed, each equipped with a set screw to adjust the tilt.

The better small mirrors and beam splitters were obtained from eBay for a few euros (although, for first explorations, simple pieces of cosmetic mirrors — or the like — will do, and a simple plane glass plate for the beam splitter). These parts were attached to the square metal brackets using cyano-acrylic glue. For the lens, I used the 50 mm lens of an old camera. Again, in this case, any lens that can accommodate the roughly ±20° divergence from the laser diode would do.

As a laser diode, a readily available 5 mW red LD can be used.

⚠ **Safety Note**: Be aware that LDs from DVD modules may produce dangerously high levels of light output! Before operating them, always read the remarks on power measurement and consult the laser safety advice for the respective laser class!

The experiments shown in this article are made with a 5 mW laser diode operated at a power level of 2...3 mW ≃ 20 mA, driven by a lab supply 0...30 V via a 1 kΩ resistor. The LD was mounted in a larger aluminum heat sink, tightly fitting in a 3.5 mm diameter hole. Thus, the temperature was kept low and reasonably stable, and no further output stabilization was necessary.

Detection Circuitry

As photodiode, a pin diode with a small detection area, such as the SFH229 by Osram, can be used. For this application, no external bias voltage is needed — see schematics in **Figure 2**.

The photodetector's electrical setup is quite simple: Its 0.3 mm^2 sensitive area is used in combination with a 100 kΩ load resistor (in this case, an applied bias would have increased the signal only marginally, and therefore it was not used).

The output of the photosensor may be monitored directly — if the oscilloscope sensitivity is high enough — or through a general-purpose amplifier, which I designed exactly for this specific need (Figure 2). It has about 200 to 300 kHz bandwidth and uses "zero"-offset op-amps.

The first stage (U1A) of this amplifier has a 20× voltage gain, whilst the second stage (U1B) works as an inverting amplifier with a 1× or 20× gain, selectable through SW1. If the (ambient) illumination is too intense, the DC coupling may lead to output saturation, and eventually a change to AC coupling may be considered. However, in these examples, DC coupling as shown in the schematics of Figure 2 works fine.

Basic Adjustment

Considerations on the interference pattern

- Focus the laser diode output on a small spot on the membrane using the photo lens.
- Place the beam splitter between photo lens and membrane with a tilt angle of about 45° so that the first reflection hits the "fixed" mirror.
- Adjust the mirror's distance such that it's the same as that of the membrane, i.e., have the laser spot also focused on the mirror.

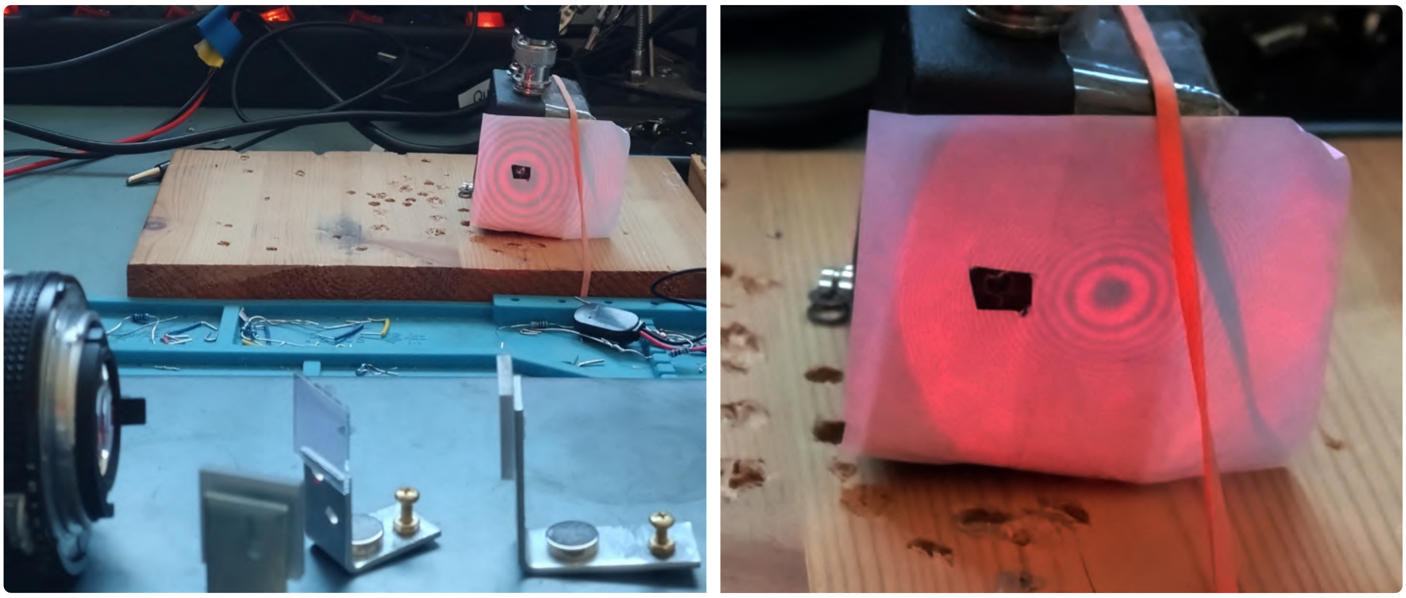


Figure 3: Aligned interference pattern with the vibrating membrane replaced by a mirror. The pattern is visible on the paper screen surrounding a hole containing the photodiode. The right side shows a situation with larger distance difference between the mirrors and off-center photodiode position. The latter can easily be fixed by moving the photodetector unit to the right.

- Tilt the fixed mirror such that the reflected intensities from the membrane (via the beam splitter reflection) and the light reflected from the mirror coincide at the photodiode.

The setup is functional if larger interference fringes are visible at the photo diode plane.

To start and get a "feeling" for the whole thing, it is a good idea to replace the investigation subject (i.e. the vibrating membrane) with another fixed mirror. That will yield better-defined (and closer to ideal) interference patterns, and reveal the issues connected with the alignment. A viable alignment configuration with both mirrors is shown in **Figure 3**.

The typical, concentric ring-shaped pattern becomes visible if both reflecting surfaces have about the same distance (path partially via beam splitter), the closer the distance match, the larger the gaps between the rings; at perfect match, only a uniform illumination with distance-dependent intensity would be visible.

For detecting an interferometer signal in the photodiode, the typical length scale of the interference pattern (in this case, the gap between rings) must be larger than the detector size of 0.56×0.56 mm). Thus, a setup as seen in the left part of Figure 3 would yield a strong, well-developed signal, while in the right part, the situation is at best marginal, and can/should be improved by moving the detector toward the ring center and improve the distance-matching between mirrors/mirror and subject under investigation.

While the situation using a good mirror in place of the vibration membrane ("the subject") yields an easily observable, quasi-ideal pattern, for real subjects, the patterns may be less clear. These membranes, etc., may not yield the perfect reflection that you get with a good mirror, but, in the case of more-or-less crumpled aluminum foil, a rugged, spotty landscape is seen.

For other items, one can also put a spot of white paint on the surface; in that case, it is particularly important that the illumination spot is in focus, i.e., that the vibrating membrane acts similarly to a light point source (with unique phase). The difference in respect to a mirror is that the intensity is not only directed in the cone defined by the illumination, but is diffused over all the possible angles; in other words, much less intensity reaches the screen/photo-diode.

In this case, the interference fringes become very difficult to recognize. However, intensity modulations upon vibration are still well observable if we use the photo-diode output AC-coupled to the oscilloscope.

Some final advice for the adjustments

- Use two mirrors to yield the concentric ring pattern on the screen such that the ring distances are maximized and position the photodiode in the center of the interference pattern.
- Remove the second mirror (without touching any of the rest) only, replace it with the vibrating membrane, etc., and adjust the distance of the membrane such that the laser focus spot has a minimal size and — in case of reflecting crumpled foil — possibly adjust the membrane orientation such that the back-reflected intensity on the photodetector is significant enough in magnitude.
- Then refine the adjustment by observing the PD-signal on the oscilloscope, while tuning the orientation and the distance of the membrane with active vibration exciter. What traces to expect are given in the following examples.

Results and Examples

A well-developed "typical" signal of a membrane with sinusoidal displacement with an amplitude of several µm is shown in **Figure 4**. With less well-defined surface reflection/adjustment, the shape may not be as ideal, as seen in **Figure 5**.

Still, in this case, and in even worse situations, the FFT spectrum clearly shows a typical maximum velocity edge.

Such an extreme situation is shown in **Figure 6**, where the plain signal, CH1, is barely visible, but the FFT spectrum still reveals the maximum intensity clearly. Here, a maximum (cutoff) frequency of about 95 kHz is quite discernible, yielding an amplitude of a ≅ 7.7 µm.

Basic Quantitative Analysis

In essence, we deal with the superposition of two light waves with a phase difference that depends on the difference, Δx, in distance these waves had to travel from a (virtual) source.

This distance is twice the membrane displacement, Δd = Δx/2, since the reflected beam must travel an extra Δd to reach the reflection point, and another Δd to return to the "null" position.

For the rest of the analysis, we assume that the displacement of the membrane is a sinusoidal:

$$\Delta d(t) = a\,sin(\omega t) + a_0$$

where some offset, a_0, is still included. The intensity, I, on the photo-diode from the superposition of the interfering beams is then:

$$I = I_1 \cos(\Delta x/\lambda) \;\; + I_0$$

I_1 is an arbitrary modulation intensity factor, and I_0 unimportant background that does not enter the AC signal.

With that, the plain AC voltage signal after amplification is

$$U(t) \propto cos([2/\lambda][a\,sin(2\pi f t) + a_0])$$

where λ is the laser wavelength, f the vibration frequency, a the membrane vibration amplitude, and a_0 an unknown displacement/phase difference.

The maximum signal frequency results from the positions with maximum phase shift rate in the cos(Δx/λ) term. This yields a contribution of cos(2/λ×2 πf t), leading to the edge frequency of the signal Fourier transform

$$f_{FFT-max} \; = \; 4\,a\,f\,\pi/\lambda$$

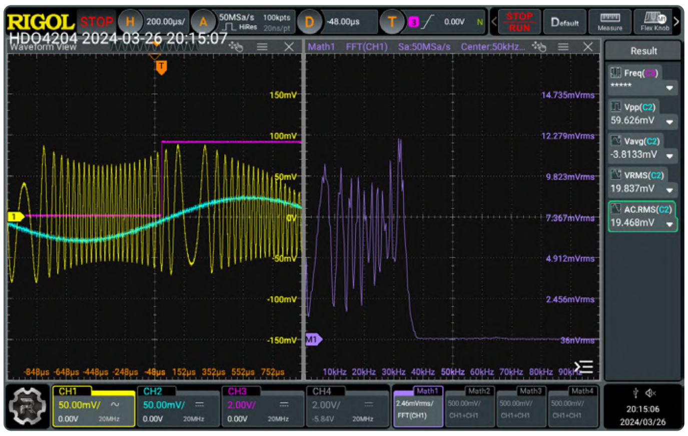

RIGOL
HDO4204 2024-03-26 20:15:07
Math1 FFT(CH1) Sa:50MSa/s Center:50kHz
Freq(C2) *****
Vpp(C2) 59.626mV
Vavg(C2) -3.8133mV
VRMS(C2) 19.837mV
AC.RMS(C2) 19.468mV
10kHz 20kHz 30kHz 40kHz 50kHz 60kHz 70kHz 80kHz 90kHz

Figure 4: Signal from a vibrating membrane. On the left: CH1 (yellow) shows the amplified photodiode output, CH2 (cyan) the exciter drive voltage (=> synchronization). Right: Fourier transform (FFT) of CH1 showing a typical shape. The high-frequency edge relates to the maximum membrane velocity.

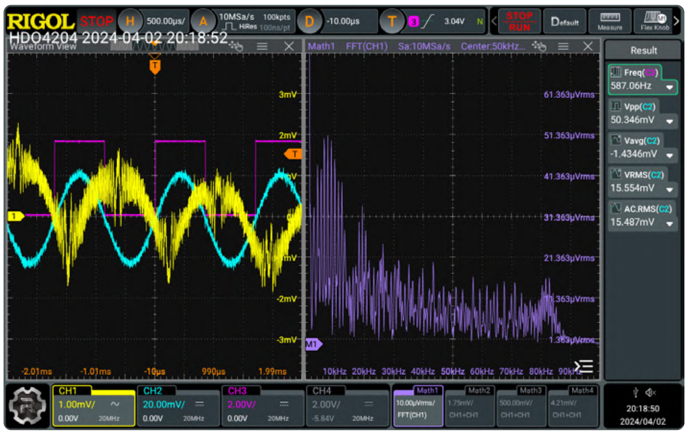

RIGOL
HDO4204 2024-04-02 20:18:52
Math1 FFT(CH1) Sa:10MSa/s Center:50kHz
Freq(C2) 587.06Hz
Vpp(C2) 50.346mV
Vavg(C2) -1.4346mV
VRMS(C2) 15.554mV
AC.RMS(C2) 15.487mV
10kHz 20kHz 30kHz 40kHz 50kHz 60kHz 70kHz 80kHz 90kHz

Figure 5: The same signal as shown in Figure 4, but from a different, less ideal setup. The traces exhibit considerable intensity modulation due to dependence of the total reflected intensity on the membrane displacement.

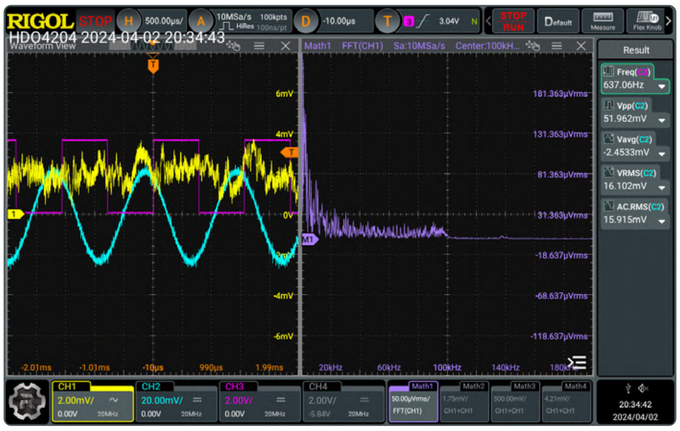
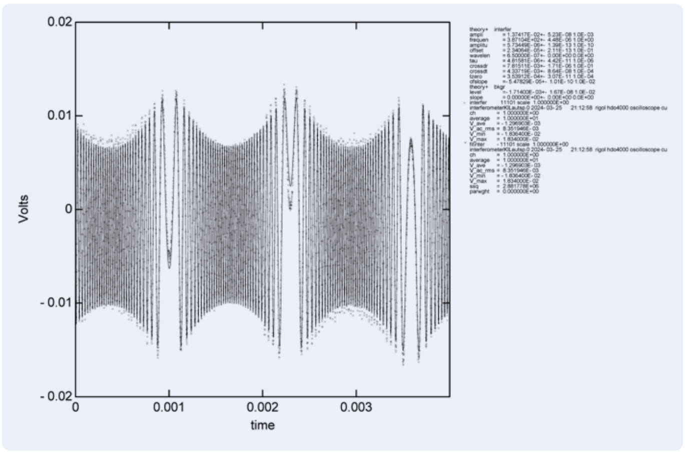
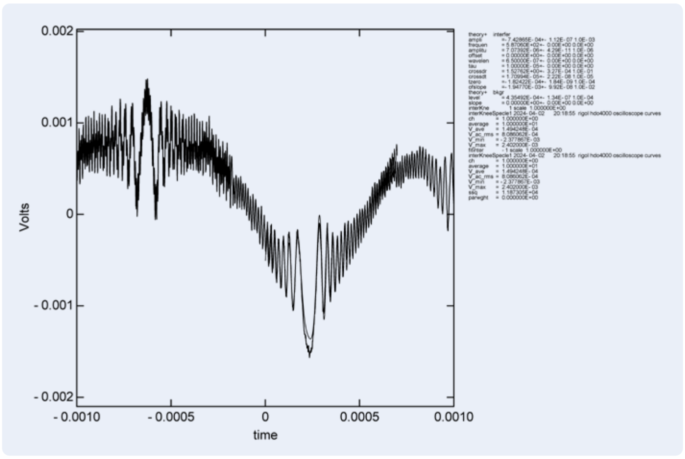

RIGOL
HDO4204 2024-04-02 20:34:43
Math1 FFT(CH1) Sa:10MSa/s Center:100kHz
Freq(C2) 637.06Hz
Vpp(C2) 51.962mV
Vavg(C2) -2.4533mV
VRMS(C2) 16.102mV
AC.RMS(C2) 15.915mV
20kHz 60kHz 100kHz 140kHz 180kHz

Figure 6: Badly defined signal from the same configuration as shown in Figure 5.



```
theory+    interfer
ampli        = 1.37417E-02+-  5.23E-08 1.0E-03
frequen      = 3.87104E+02+-  4.48E-06 1.0E+00
amplitu      = 5.734496E-06+-  1.39E-13 1.0E-10
offset       = 2.34064E-05+-  2.11E-13 1.0E-01
wavelen      = 6.50000E-07+-  0.00E+00 0.0E+00
tau          = 4.81581E-06+-  4.42E-11 1.0E-06
crossdr      = 7.81511E-03+-  1.71E-06 1.0E-01
crossdt      = 4.33719E-03+-  8.64E-08 1.0E-04
tzero        = 3.53912E-04+-  3.07E-11 1.0E-04
ofslope      =- 5.47829E-05+-  1.01E-10 1.0E-02
theory+    bkgr
level        =- 1.71400E-03+-  1.67E-08 1.0E-02
slope        = 0.00000E+00+-  0.00E+00 0.0E+00
interfer        11101 scale  1.000000E+00
interferometerKILautsp.0 2024-03-25     21:12:58  rigol hdo4000 oscilloscope cu
ch         =  1.000000E+00
average    =  1.000000E+01
V_ave      = - 1.296903E-03
V_ac_rms   =  8.351946E-03
V_min      = - 1.836400E-02
V_max      =  1.834000E-02
fitInter      - 11101 scale  1.000000E+00
interferometerKILautsp.0 2024-03-25     21:12:58  rigol hdo4000 oscilloscope cu
ch         =  1.000000E+00
average    =  1.000000E+01
V_ave      = - 1.296903E-03
V_ac_rms   =  8.351946E-03
V_min      = - 1.836400E-02
V_max      =  1.834000E-02
ssq        =  2.881778E+06
parwght    =  0.000000E+00
```

Figure 7: Fit of an observed interferometer signal with the full theoretical model, yielding an amplitude of 5.735E-6 m (5.736 µm). Besides noise fluctuations, the computed curve (line) meets the data points (circles) quite well.



```
theory+    interfer
ampli        =- 7.42865E-04+-  1.12E-07 1.0E-03
frequen      = 5.87060E+02+-  0.00E+00 0.0E+00
amplitu      = 7.07392E-06+-  4.29E-11 1.0E-06
offset       = 0.00000E+00+-  0.00E+00 0.0E+00
wavelen      = 6.50000E-07+-  0.00E+00 0.0E+00
tau          = 1.00000E-05+-  0.00E+00 0.0E+00
crossdr      = 1.52762E+00+-  3.27E-04 1.0E-01
crossdt      = 1.70994E-05+-  2.22E-08 1.0E-05
tzero        =- 1.82422E-04+-  1.84E-09 1.0E-04
ofslope      =- 1.94770E-03+-  9.92E-08 1.0E-02
theory+    bkgr
level        = 4.35492E-04+-  1.34E-07 1.0E-04
slope        = 0.00000E+00+-  0.00E+00 0.0E+00
interKne          1 scale  1.000000E+00
interKneeSpecle1 2024-04-02     20:18:55  rigol hdo4000 oscilloscope curves
ch         =  1.000000E+00
average    =  1.000000E+01
V_ave      =  1.494248E-04
V_ac_rms   =  8.086062E-04
V_min      = - 2.377867E-03
V_max      =  2.402000E-03
fitInter        - 1 scale  1.000000E+00
interKneeSpecle1 2024-04-02     20:18:55  rigol hdo4000 oscilloscope curves
ch         =  1.000000E+00
average    =  1.000000E+01
V_ave      =  1.494248E-04
V_ac_rms   =  8.086062E-04
V_min      = - 2.377867E-03
V_max      =  2.402000E-03
ssq        =  1.187305E+04
parwght    =  0.000000E+00
```

Figure 8: Fit in a case with large crosstalk contribution intensities (fit: smooth line over the time interval -0.0002 to 0.001 s, noisy data line).

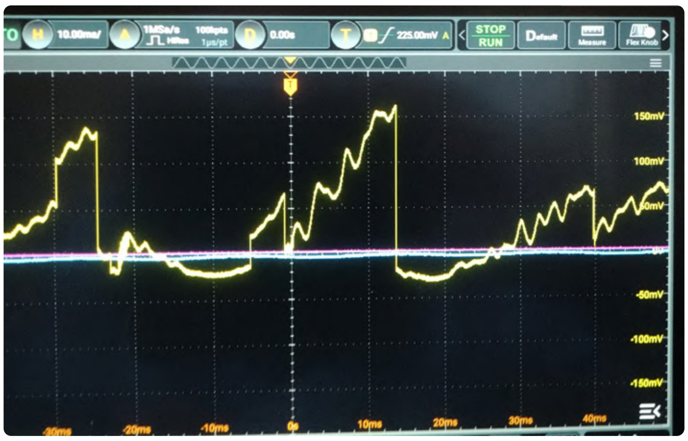


Figure 9: Interferometer signal showing unwanted mode jump effects.

which then can be used to infer the vibration amplitude as

$$a \;=\; f_{FFT-max}\,\lambda\,/(4\pi f)$$

The example in Figure 5 displays a limit frequency $f_{FFT\text{-}max} \simeq 35$ kHz with the laser wavelength of 650 nm, and drive frequency f ≃ 800 Hz, so we estimate a ≃ 2.3 µm.

More Math for the Nerds

In Figure 4, with the highest oscillation frequency, you may have noticed the drop of oscillation amplitude in the center. This reduction stems from the implicit low-pass filtering of the photodetector-amplifier chain. Further, Figure 5 exhibits the influence of (non-interference related) intensity modulation due to membrane vibrational movements.

If we want to describe the observed waveforms exactly and eventually determine their parameters by fitting, the mathematical description must be carried a bit beyond the expression for U(t) given above.

Note: On the Elektor Labs page for this article [2], the core routine with small programs to compute the intensity function U(t) (up to a scaling factor in front of it) are supplied in FORTRAN or C. Both versions are doing basically the same thing. Quick instructions for running them are also included.

First, the limited bandwidth is accounted by

$$\hat{U}(t) = \frac{1}{\tau}\int_0^{10\tau} U(t-u)e^{-u/\tau}\,du$$

where τ = RC is the effective low-pass filtering time constant.

Second, the crosstalk of vibration-to-illumination intensity is added:

$$\bar{U}(t) \;=\; \hat{U}(t) \;+\; U_{crosstalk}\,\sin(2\pi f\,[t+t_c])$$

Further, we allow for an additional linear drift of $a_0 = a_{00} + a_{0t}t$.

With these additions, it is indeed possible to fit the observed interferometer intensity curves nicely, as is illustrated in **Figure 7** and **Figure 8**.

It is, however, a tedious procedure to find the proper starting values, such that the fit locks into the proper minimum.

Depending on excitation amplitude, amplitudes between 2 and 10 µm were observed (with an arrangement similar to the E-field sensor contraptions). This is reasonably in concordance with the 2 µm average and 20 µm maximum amplitude inferred from the E-field sensor signals.

Laser Diode, Operation, BPW34 as Power Meter

For a red laser diode (λ ≃ 650 nm), the sensitivity of BPW34 can be inferred from the data sheet as being about 0.3 A/W (or 0.3 mA/mW).

This current value can be measured with any decent multimeter directly — we don't even need, necessarily, further resistors or bias voltages. For a (safe) estimate of laser diode power, it is sufficient to connect the photodiode (BPW34, in our case) to the multimeter (switched to mA range), and then illuminate the diode.

The only condition to be met is that all the light output must hit the sensitive area of the photodiode. By juxtaposing LD and PD at contact, this can easily be achieved. (Possibly, move the relative positions to find the maximum). Then simply multiply the mA reading by 3.3 to get an estimate of the LD output in mW.

Caveat: You may encounter photodiode fluctuations that are connected to laser mode jumps, rather than membrane motions. These may result from drifts in the laser diode temperature and/or light backfeeding into the diode.

Fortunately, it is easy enough to discriminate backfeed from the "genuine" interferometer signals in these situations. Whilst the genuine signal is smooth and resembles sinusoidal shapes as shown in Figure 4, the mode jumps are sudden and visible as jumps in the PD output, as seen in **Figure 9**.

240279-B-01

Questions or Comments?

Do you have technical questions or comments about this article? Please contact the author at michael.monkenbusch@googlemail.com or the Elektor editorial team at editor@elektor.com.

About the Author

Michael Monkenbusch is a retired physicist who worked in the fields of neutron scattering, instrumentation, and soft-matter physics. Reviving an old electronics hobby led to the project presented here.

WEB LINKS

[1] Elektor Labs page for the E-Field Measurement Project: https://tinyurl.com/emfmeasurement
[2] Elektor Labs page for the Laser Vibrometer: https://tinyurl.com/laservibrometer

Crystals and Oscillators

Improving Crystal Accuracy Through Capacitor Selection

By Susanna Engel Rodrigues (Würth Elektronik eiSos)

Almost every electronic circuit relies on a timer, typically driven by a crystal to generate precise clock signals. Crystals supply the required clock signals. The load capacitors in oscillator circuits with crystals play a key role, as they have a significant influence on accuracy and reliability.

Electronic devices permeate our everyday lives, making numerous activities easier and providing entertainment in our leisure time. Hardly any electronic application can do without microcontrollers and other digital components that have one thing in common — they need an accurate and reliable clock signal. This is where crystals and oscillators come into play.

Crystals ensure the correct clock frequency in a microcontroller circuit and are therefore essential for it to function properly. However, to ensure that this is the case, there are a few things to consider that may seem simple at first glance, such as the correct choice of capacitors for the crystal circuit.

Circuitry of Quartz Crystals

Crystals have been used for many decades to generate clock signals. However, as quartz crystals are passive components, a so-called activation circuit is always required to turn the passive quartz crystal into an active oscillator. The most common variant of this oscillator circuit is the *Pierce oscillator* as shown in **Figure 1**. In most cases, the inverter (Inv) and the resistor R_f are already integrated in the microcontroller. The two capacitors C_a and C_b, on the other hand, must be added and are an integral part of the load capacitance.

Key Parameter Load Capacitance

The load capacitance is one of the most important parameters in the specification of a quartz crystal. Often referred to as C_L or C_{Load}, in crystal datasheets, this parameter is often misunderstood or misinterpreted.

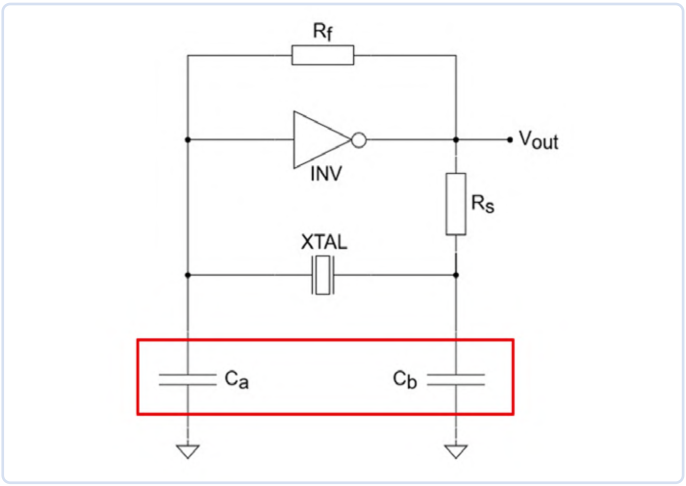


Figure 1: Pierce oscillator with focus on the load capacitors C_a and C_b.

One misconception, for example, is that the value represents the capacitance of the crystal. Another is that it directly indicates the values for the required capacitors C_a and C_b. However, this value indicates the load at which the crystal was tuned to its target frequency during production. Consequently, a crystal will only oscillate at its target frequency, if the circuit in which it is installed, also supplies this load capacitance. Ideally, the following applies:

Load Capacity Quartz Datasheet = Load Capacity Circuit

The following simple formula can be used to correctly design the load capacitance in the circuit:

$$C_L = \frac{C_a * C_b}{C_a + C_b} + C_{stray}$$

However, this formula has a few pitfalls when it comes to implementation.

The stray capacitance C_{stray} includes both the capacitance of the traces as well as the capacitance of the input and output pins of the microcontroller to which a crystal is connected. While the capacitance of the microcontroller pins is occasionally specified, that of the traces and thus the total value of the stray capacitance usually remains unknown. This means that the value of the stray capacitance can only be estimated during the initial design of the circuit. For the PCB layout, it is important to bear in mind that the stray capacitance of the traces is largely determined by their length — the shorter the traces, the lower the stray capacitance.

For the selection of the two capacitors C_a and C_b according to Figure 1, it has proven useful in practice to assume the same values ($C_a = C_b$). This not only simplifies the calculation and purchasing, but typically provides good results. If this is not suitable due to the stray capacitance and the fixed capacitance values of E-series capacitors, the capacitor with the larger capacitance should be used in place of C_b with $C_a < C_b$.

Selection of the Capacitor Type

Normally, MLCCs (multilayer chip capacitors) are used. When selecting the capacitors, care should be taken to ensure that they have the most stable capacitance over temperature and DC bias behavior possible. Ideally, capacitors of type NP0, such as the WCAP-CSGP General Purpose series [1] or the WCAP-CSRF High Frequency variant [2]. Otherwise, temperature fluctuations can also have a negative impact on an otherwise well-tuned oscillator design.

Easy options for selecting capacitors are offered by design platforms such as RedExpert [3] from Würth Elektronik.

Consequences of Incorrectly Designed Load Capacitance

If the circuit's load capacitance is not correctly specified, it will directly influence the frequency tolerance of the quartz crystal. The extent of this influence is indicated by the so-called trim sensitivity T_S:

$$T_S\left[\frac{ppm}{pF}\right] = \frac{C_1 * 10^6}{2(C_0 + C_L)^2}$$

The trim sensitivity of a quartz crystal depends on the size, blank shape and frequency, as these define the geometry of the quartz blank and therefore the values of the dynamic capacitance C_1 and the shunt capacitance C_0.

Figure 2 illustrates the trim sensitivity resulting from incorrect load capacitance. Three difference crystal designs were selected here as examples. Apart from the size, the specifications are identical in terms of frequency (24 MHz), tolerance (±10 ppm) and load capacitance (8 pF). The curves show the trim sensitivity. If the load capacitance corresponds to the circuit of the quartz (i.e., 8 pF), there is no additional frequency deviation: the quartz crystal oscillates at its target frequency.

However, if the load capacitance of the circuit is higher, 10 pF for example — there is a frequency shift of -10 to -36 ppm depending on the crystal. If the capacitance is lower, 6 pF for example, the influence is even greater, leading to deviations of up to 56 ppm. In the 24 MHz example, this corresponds to an additional frequency offset of 1344 Hz.

It is also clear that the trim sensitivity is not a linear function, but rather has a steeper curve with decreasing load capacitance, i.e., a greater frequency deviation. This should be considered when selecting a quartz crystal: The lower the load capacitance, the more susceptible a crystal is to deviations.

These large deviations in the clock frequency can, in turn, have far-reaching consequences for the application:

- Unstable or unpredictable behavior of the microcontroller, for example, restarting the microcontroller

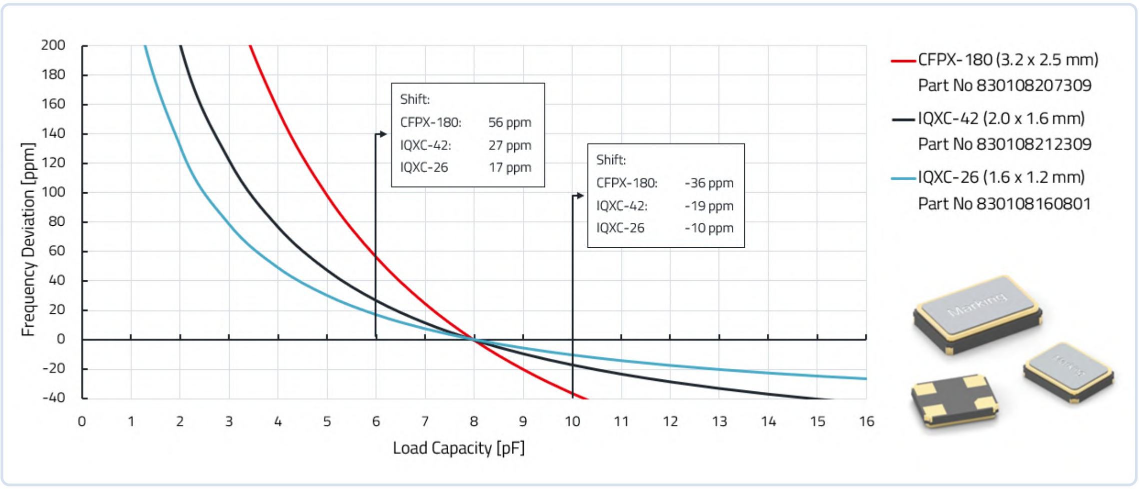


Figure 2: Trim sensitivity of different sizes.

- Malfunctions and, as a result, data loss or communication errors
- Impairment of the performance of the overall system
- Total failure

Checking the Load Capacitance

The only way to determine whether the load capacitance has been designed correctly is to measure the frequency. This is done either at an isolated output (some microcontrollers offer this option directly) or with a special probe with very low capacitance. If a normal probe is used for measuring in the circuit, its load already influences the frequency, and the measurement is no longer meaningful.

The deviation can be easily calculated (in ppm) according to the measured frequency compared to the target frequency according to the datasheet:

$$Frequency_{deviation} = \left(\frac{Frequency_{measurement}}{Frequency_{datasheet}} - 1\right) * 10^6$$

If the deviation is greater than the permissible tolerance, it can be readjusted accordingly:

- $Frequency_{deviation} > 0$ ⇒ Increase of the capacitance values necessary
- $Frequency_{deviation} < 0$ ⇒ Reduction of the capacitance values necessary

Power Consumption and PCB Design

Despite a higher trim sensitivity, the choice of a quartz crystal with a low load capacitance offers the advantage of lower power consumption. In battery-operated applications, this can be a decisive factor for a low target load capacitance. In addition, the lower the load capacitance, the higher the amplification and thus the safety factor in the oscillator circuit.

When laying out the PCB, as mentioned above, care should be taken to ensure that the conductor paths between the microcontroller, capacitors and crystal are kept as short as possible. This reduces the stray capacitance and shorter lines are always preferable in terms of radiation and coupling of interference signals. Furthermore, the trace lengths should ideally be the same. If this is not possible, it is advisable to place C_a closer to the microcontroller to further reduce stray capacitance.

A good ground connection is important — ideally by means of a ground plane only for the crystal and its peripherals, which is connected to a separate GND pin on the microcontroller and thus separates the crystal from the rest of the circuit.

Design Kits and Integrated Oscillators

Although crystals are often regarded as simple components that are easy to integrate into a circuit, a closer look shows that a crystal can only oscillate as well as the associated circuit design allows. There are some potential pitfalls, but with a little effort, they can be avoided so that the circuit remains accurate and reliable.

Design Kits such as the 830004 [4] from Würth Elektronik offer everything from a single source for development — here you will find crystals in common frequencies and load capacities with appropriate capacitors and corresponding design recommendations.

However, if you find this too much effort, you can also fall back on a ready-made solution: Integrated oscillators, often referred to as XOs or SPXOs [5] (Simple Packaged Xtal Oscillators), already integrating all the necessary components and only need to be connected to a supply voltage and the microcontroller.

250566-01

About the Author

Susanna Engel Rodrigues completed a dual study program in Project Engineering at the DHBW Mosbach and graduated with B.Eng. and B.Sc. While working, she also completed an M.Sc. in International Business Management at the University of Leeds. Susanna Engel Rodrigues has been working in the field of frequency products since 2006 and has been a Field Application Engineer at Würth Elektronik eiSos since 2018. Her experience ranges from crystal blank grinding to the final installation of the frequency product.

WEB LINKS

[1] Multilayer ceramic capacitors of the WCAP-CSGP MLCC series, Würth Elektronik: https://tinyurl.com/mlcc-general-purpose
[2] High-frequency multilayer ceramic capacitors of the WCAP-CSRF High Frequency series, Würth Elektronik: https://www.we-online.com/en/components/products/WCAP-CSRF
[3] Online design platform RedExpert, Würth Elektronik: https://redexpert.we-online.com/we-redexpert/en/#/home
[4] Design kit quartz crystals & matching capacitors 830004, Würth Elektronik: https://www.we-online.com/en/components/products/DESIGN_KIT_830004
[5] Quartz oscillators of the WE SPXO series, Würth Elektronik: https://www.we-online.com/en/components/products/WE-SPXO
[6] Würth Elektronik webinar on YouTube: "Keeping the rhythm: how capacitor selection affects crystal accuracy": https://www.youtube.com/watch?v=ST5D3oPkGoA

Starting Out in Electronics...

Special Audio ICs

By Eric Bogers (Elektor)

With the previous installment, we have essentially concluded the topic of operational amplifiers and are gradually approaching the end of this series of articles. However, before we reach that point, we will first turn our attention to special audio ICs.

Here, we examine several special ICs that are commonly used in audio technology. These special-purpose ICs allow most (audio) tasks to be carried out in a very straightforward manner and with a minimum of external components.

Important: the text on which this article series is based dates back to 2005. Naturally, this has implications for the availability of the ICs discussed below. As far as we have been able to verify, all ICs featured here are still available at the time of writing (May 2025); however, no guarantees can be given. Nonetheless, the descriptions below will provide sufficient guidance to help you identify suitable equivalents if necessary.

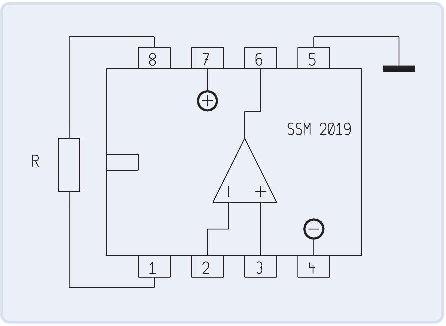


Figure 1: The SSM2019.

Microphone Preamplifier SSM2019

The SSM2019 from Analog Devices (**Figure 1**) is an extremely low-noise preamplifier. The noise generated by the SSM2019 is only 1.5 dB above the absolute minimum dictated by physical laws, and distortion is also remarkably low at 0.01% with a gain factor of 100 across the entire audio range.

In terms of pin configuration, the SSM2019 resembles a standard single operational amplifier. However, pin 5 of the SSM2019 must be connected to ground, while the gain is set using a resistor between pins 1 and 8.

Table 1: External Resistor and Gain (SSM2019)

R (Ω)	Gain	Gain(dB)
–	1	0
4k7	3.15	10
1k1	10	20
330	31.5	30
100	100	40
32	315	50
10	1000	60

Table 1 shows the relationship between the resistance value and the gain factor.

The common-mode rejection ranges from 54 dB at unity gain to 112 dB at a gain of 1000 and remains relatively constant across the entire frequency range.

Line Receiver SSM2143

In the not-so-distant past, signal transformers were essentially the only components used for balanced inputs. They offered not only the advantage of galvanic isolation but also provided excellent common-mode rejection, significantly mitigating the adverse effects of induced interference.

High-quality audio transformers, however, are expensive, which has led to the increasing use of electronic balancing as an alternative. Several such circuits have been discussed in a previous installment under the headings “differential amplifier” and “instrumentation amplifier.” However, these circuits typically exhibit only modest common-mode rejection at low gain settings. This limitation is primarily due to the unavoidable tolerances of the resistors used. Metal film resistors generally have a tolerance of 1%, making it largely a matter of chance to achieve a common-mode rejection exceeding 40 dB.

In the SSM2143 (**Figure 2**), the relevant resistors are already integrated into the chip and are laser-trimmed during the manufacturing process. As a result, this IC achieves a common-mode rejection between 90 dB (at 50 Hz) and 85 dB (at 20 kHz) —values that can confidently stand comparison with

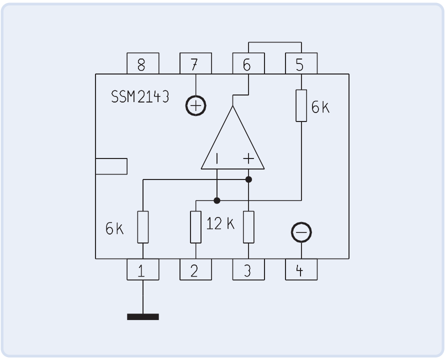


Figure 2: The SSM2143.

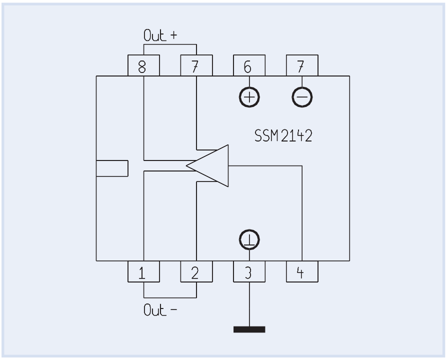


Figure 3: The SSM2142.

those of high-quality signal transformers. It should be noted, however, that galvanic isolation is not provided.

The circuit shown in Figure 2 is a differential amplifier, as previously discussed in this article series. Due to the resistor configuration, the gain is -6 dB, meaning that the standard studio-level signal of +6 dB is reduced to 0 dB, which corresponds to the typical internal signal level in electronic equipment

Line Driver SSM2142

A line driver is naturally, the counterpart of a line receiver, and is used to convert a signal into a balanced format. With a gain of 6 dB, it amplifies the internal signal level of the mixing console to the studio standard of +6 dB. Thus, the SSM2142 (**Figure 3**) serves as the ideal complement to the SSM2143 discussed in the previous section.

Achieving good common-mode rejection is generally more difficult in output stages than in input stages; consequently, the SSM2142 provides a common-mode rejection of "only" 45 dB. The IC is capable of driving loads of 600 Ω or higher with an output voltage of 10 V_{RMS}. Pins 2 and 7 are sense inputs that must be connected to the corresponding outputs. (Refer to the datasheet for detailed guidance.)

VCA THAT2150

To construct compressors and gates, voltage-controlled amplifiers (VCAs) are required. In many mixing consoles, VCA groups serve as a supplement to the existing subgroups. The VCA produced by THAT Corporation (**Figure 4**) has become something of a standard in professional audio engineering.

The THAT2150 is housed in a SIL package (Single In Line, with all pins in a single row). To minimize noise contribution, the IC features current inputs and outputs; therefore, a series resistor is required at the input and an operational amplifier must be used at the output.

The attenuation is controlled by the voltage applied to pin 3, with a rate of 1 dB per 6 mV. This means that to achieve an attenuation of 100 dB, a control voltage of 600 mV must be applied. If the signal needs to be amplified rather than attenuated, a negative control voltage is used, or alternatively, the non-inverting input of the IC (pin 2) is employed.

According to the datasheet, the IC offers a maximum attenuation of 130 dB, a dynamic range of 116 dB, and a total harmonic distortion of 0.008%. To achieve this low distortion figure in practice, the symmetry must be adjusted using the trimmer potentiometer shown in Figure 4.

In combination with the RMS level detector THAT2252, a compressor can be implemented without major difficulty — but this becomes even simpler with the THAT4301 (see the following section), which is inherently equipped with all the necessary subcircuits.

THAT4301

The THAT4301 integrates a VCA (essentially equivalent to the THAT2150 described above), an RMS level detector, and three low-noise operational amplifiers within a single package — one of which is internally connected to the VCA output. The remaining two op-amps can be freely configured by the user.

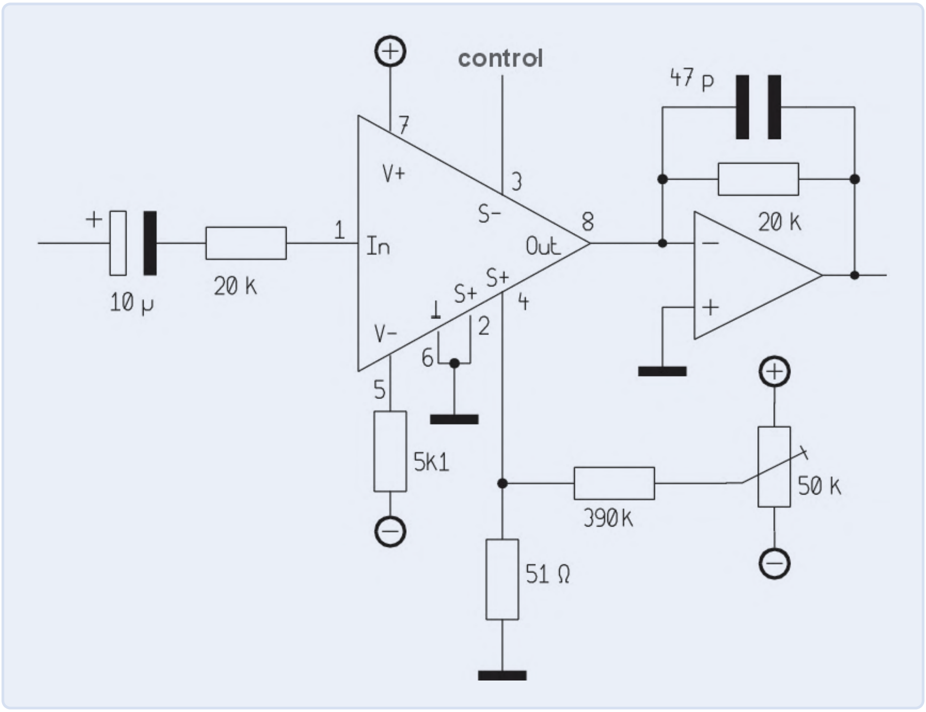


Figure 4: Basic circuit with the THAT2150.

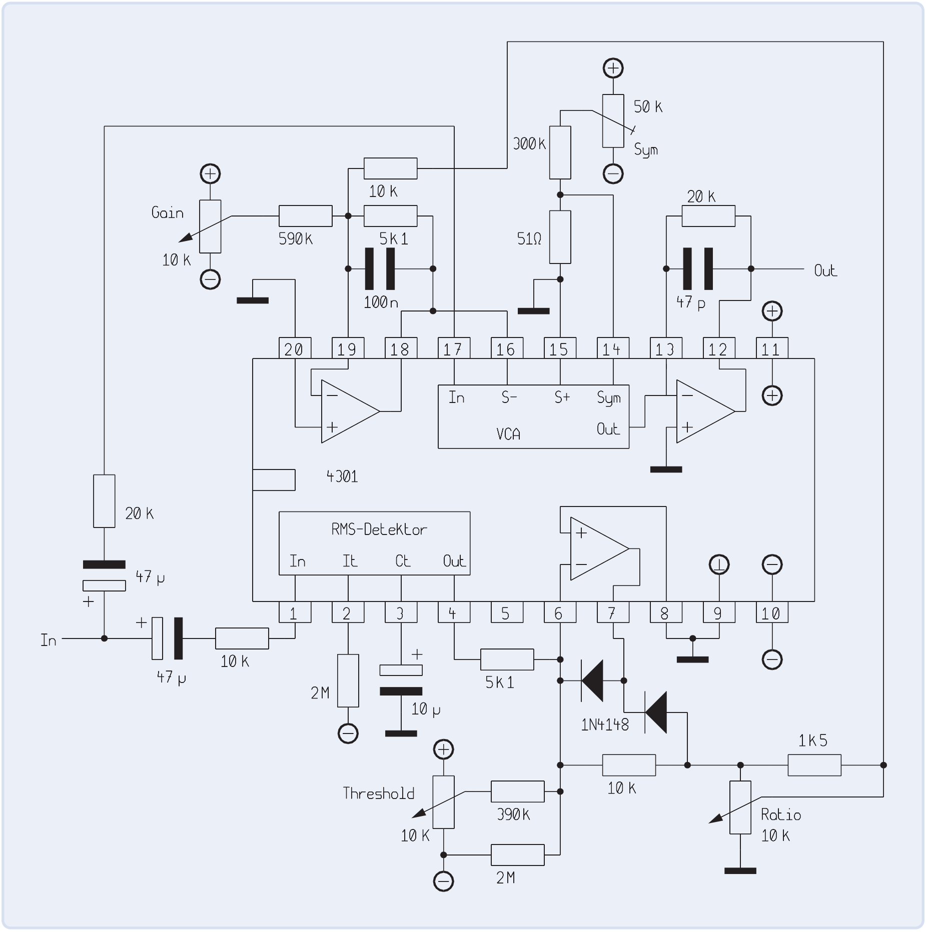


Figure 5: Simple compressor using the THAT4301.

The circuit shown in **Figure 5** illustrates a simple yet high-quality compressor. The control elements are limited to just three knobs: *threshold*, *ratio* and *gain*.

Let us follow the signal path through the circuit. From the input, the signal is routed to the VCA, which, like its discrete counterpart, features a current input and a current output. A series resistor is therefore required at the input, while an operational amplifier is needed at the output — this is already internally connected to the VCA. For optimal performance and minimal distortion, the symmetry of the circuit must be adjusted.

Now consider the control section of the circuit. This also involves a current input. The resistor at input It (pin 2) and the capacitor at input Ct (pin 3) determine the time behavior of the RMS level detector. The It current sets the level (more precisely, the input current) at which the detector produces an output voltage of 0 V. Although the exact value is less critical, what does matter is that the RMS detector provides a voltage corresponding to the required control signal for the VCA, specifically 6.5 mV/dB.

The operational amplifier following the detector is configured as an inverting precision rectifier with a summing input. The threshold control adjusts the input signal to the detector slightly up or down. When the signal exceeds 0 V, the rectifier outputs the same value but inverted; otherwise, the rectifier blocks the signal.

Thanks to the 5.1 kΩ and 10 kΩ resistors, the circuit provides a gain of approximately 2. This means that for every dB the input signal exceeds the threshold, the output becomes 13 mV more negative. This gain is then reversed by the following inverting amplifier.

If this control signal were used directly to drive the VCA, the result would effectively be a limiter, since each 13 mV increase would reduce the gain by 1 dB. To achieve a lower compression ratio, the control voltage is divided via the ratio control. For instance, halving the control voltage yields a compression ratio of 2:1.

The 1.5-kΩ resistor provides a more practical control range: without it, compression ratios from 1:1 to 2:1 would consume half of the available adjustment range. With the added resistor, setting the ratio control to its center position yields a compression ratio of approximately 5:1.

The next stage has a gain of 0.5, effectively canceling out the voltage doubling introduced by the previous stage. Without this attenuation, the compressor would behave as a 1:1 system when the ratio control is fully engaged, meaning that for each dB exceeding the set threshold, the output signal would drop by 1 dB below the threshold. This amplifier stage is also configured as a summing amplifier: the voltage from the gain control is added to the control signal at this point.

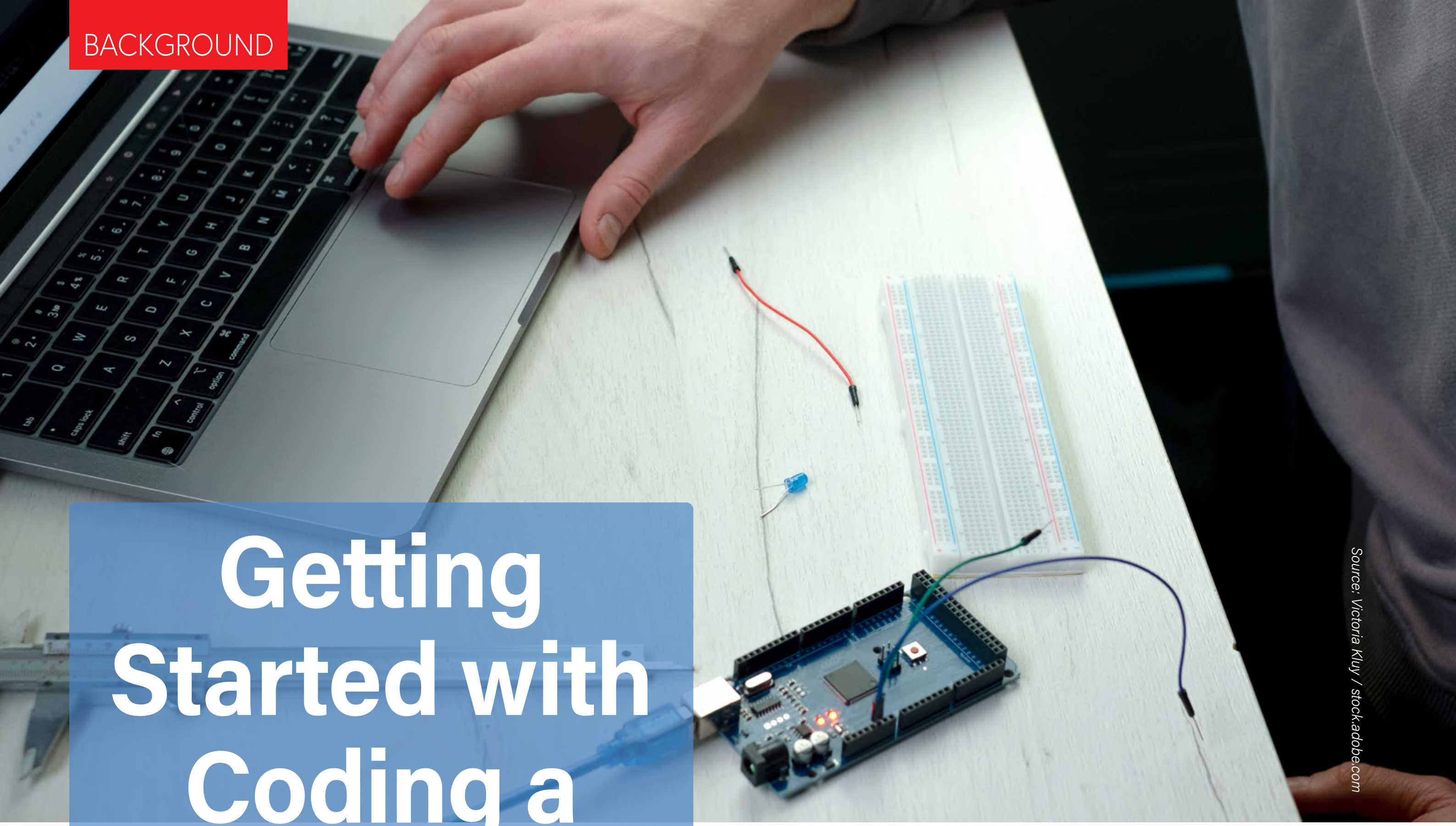
We will conclude this installment at this point; in the next issue, we will address several power supply-related topics.

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.*

250373-01

Questions or Comments?

Do you have technical questions or comments about his article? Send an email to Elektor at editor@elektor.com.



Getting Started with Coding a DIY Project

By Michael Parks (Mouser Electronics)

It is increasingly rare to build electronic systems that consist only of hardware. In both professional and hobbyist markets, embedded systems are a marriage of passive and active electronic components and software or firmware.

A quick aside: The term firmware originates from the late 1960s and is a combination of "firm" (suggesting something solid but not as rigid as hardware) and "ware" (from software). Firmware is "firm" in the sense that it is stored in non-volatile memory (like ROM, EPROM, or flash memory) and is essential for a device's operation. However, it can still be updated or modified.

So, what does it take to develop code for DIY electronics, specifically microcontroller-based projects? First, let's examine the embedded programming languages geared toward hobbyists. The most common options are Arduino (C/C++), MicroPython, CircuitPython, and their associated runtime environments.

- **Arduino (C/C++)**: The Arduino platform provides a simplified version of C/C++ that abstracts low-level complexities while offering supporting libraries for embedded development. Its advantages include better performance for critical applications, lower memory usage, greater hardware compatibility, broader peripheral support, and more control over low-level hardware.
- **MicroPython** and **CircuitPython**: These implementations of the Python language were created to address some of the challenges of C/C++ programming. Advantages include simplified library management, interactive debugging, automatic storage mounting, and better error handling.

If you have experience in desktop programming in C/C++ or Python, you should pick the embedded counterpart to reduce the learning curve. For those without programming experience, reviewing sample projects in each language can help determine which feels most natural. In general, C/C++ offers better performance, while Python-based options prioritize ease of use and readability.

Steps for Writing Firmware

Regardless of the hardware platform and programming language selected, the high-level process of writing code is essentially the same. First, we must get our logic out of our brains and into the computer. This can be achieved in a simple text editor like Notepad or a more advanced editor like Visual Studio Code. Simply write down the steps your code needs to walk through in a numbered list. We aren't worried about coding yet; we are just documenting how we envision our code will flow. What inputs are taken in, how are we processing the data, and what outputs do we generate and when? This is called pseudocode and is written in your native language.

After we have documented the high-level requirements for our code, it's time to write the firmware. The code we produce in human-readable format is called source code. The file extension of a source code file typically indicates the programming language used. For example, *.ino* and *.c* are indicative of Arduino and C programs, whereas *.py* and *.mpy* are the file extensions of Python-based programs. You can write source code in a simple text editor. However, novices might consider an integrated development environment (IDE).

An IDE is a software application that provides a comprehensive set of tools for software development in one package. In addition to a code editor, it also includes useful tools such as a compiler/interpreter, debugger, and version control. While it is possible to manually set up the toolchain that allows one to compile source code into a file understood by a microcontroller (i.e., machine code) and then upload it to the device using specialized hardware, an IDE handles all this complexity in a single application from the user's perspective.

Conversely, MicroPython and CircuitPython are interpreted languages. This means the source code remains in a human-readable format and is executed directly by the Python interpreter on the microcontroller. This removes the need for a compilation step but requires the microcontroller to have sufficient resources to run the interpreter in addition to the source. The CircuitPython firmware must first be flashed to the hardware. After a reboot, the development board will appear as a USB mass storage device and the user's source code can be copied to the apparent flash drive. After another reboot, the microcontroller will begin executing the code.

Additional Tips for Success

Whether you use compiled code, such as Arduino, or interpreted code, like CircuitPython, it is important to remember to wire your external hardware before uploading firmware. Ensure the USB cable you use to connect your microcontroller development kit to your development board carries data. If you connect your board and onboard LEDs light up, but your IDE doesn't see the board, you may have incorrect drivers or are using a power-only USB cable that lacks data lines. Also, consider the following when getting started with coding for your DIY projects:

Start with Simple Projects

Blinking an LED, reading a sensor, or controlling a motor are great beginner exercises. They help you learn how software influences hardware and vice versa. It is not always intuitive, so start small and build confidence.

Remember to Debug

Code will rarely, if ever, work as intended on the first try. Developing debugging skills is crucial. Professional engineers may rely more on hardware debugging tools, such as hardware debuggers, oscilloscopes, and logic analyzers, to inspect signals, particularly when building custom circuit boards or interfacing with many external components. However, a simple way to debug for beginners is to use the serial terminal to pass messages from the microcontroller to the host machine. The Arduino ecosystem uses the `serial.print()` function for this purpose. Similarly, in MicroPython, the `print()` function outputs messages via the read-eval-print loop (REPL) interface. That said, even low-cost development kits incorporate built-in debugging hardware so that users can insert breakpoints and peek into registers via the IDE.

Explore Third-Party Libraries, But Also Write Your Own

If you are considering adding functionality or external components (e.g., sensors, actuators) to your project, chances are that code already exists to provide the interfaces. Arduino libraries and MicroPython modules are great ways to get a project working fast and reliably. However, overreliance on libraries and modules may deprive you of a great learning opportunity to understand the intersection of hardware and software. Once you get a project up and running, consider forking a library that interfaces a sensor and the microcontroller and see if you can add functionality that is not native to the library. Reading the library or module code is beneficial as you will have to interact more directly with the hardware (e.g., ports and registers).

Experiment and Learn

Hands-on learning and iterative improvements will enhance your understanding of microcontroller programming. Beyond the basic general-purpose input/output (GPIO), learn how to interact with communication protocols (e.g., I²C, SPI, UART), power management, task scheduling, and other advanced features your hardware supports. Pick a new feature of your microcontroller and learn how to write the code that controls that functionality.

It Has Never Been So Easy

Getting started with embedded programming has never been more accessible. Platforms like Arduino, MicroPython, and CircuitPython provide powerful options regardless of your technical background. Arduino, built on C/C++, delivers high performance and precise hardware control, while Python-based environments prioritize ease of use and rapid development. Writing firmware is a structured process: Start with pseudocode, move to an IDE to program the source code, and then test with real hardware interactions and serial output. Beginners should start with fundamental projects like blinking an LED before diving into debugging techniques, serial communication, and built-in diagnostic tools. While third-party libraries accelerate development, writing custom code deepens understanding and enhances flexibility. Exploring core microcontroller functions like I²C, SPI, and UART opens the door to more advanced designs, helping engineers push the limits of what's possible.

250553-01

About the Author

Michael Parks, P.E., is a contributing writer for Mouser Electronics and the owner of Green Shoe Garage, a custom electronics design studio and technology consultancy located in Southern Maryland. He produces the STEAM Power Podcast to help raise public awareness of technical and scientific matters. Michael is also a licensed Professional Engineer in the state of Maryland and holds a Master's degree in systems engineering from Johns Hopkins University.

SPECTRAN® V6 Mobile

Modular, Configurable Real-Time Spectrum Analyzer for Reliable Measurements Across All Frequency Ranges

Contributed by Aaronia AG

Increasing channel bandwidths in mobile networks demand new solutions for frequency monitoring and signal analysis. A significant increase in the real-time bandwidth (RTBW) of measurement equipment plays a crucial role here. Additionally, modern infrastructures require portable real-time spectrum analyzers with large storage capacities. Aaronia AG has met these challenges by developing the SPECTRAN® V6 Mobile series — highly customizable, extremely fast, and reliable USB real-time spectrum analyzers.

Wireless communication is undergoing a profound transformation with the rollout of 5G and the path toward 6G. In this highly dynamic environment, traditional spectrum analyzers are often no longer sufficient. Real-time spectrum analyzers (RTSAs) have become essential tools in the development, testing, and operation of modern wireless networks.

Traditional spectrum analyzers typically use swept-tuned technology, scanning the frequency spectrum sequentially. While adequate for static or low-dynamic scenarios, this approach falls short in modern mobile networks, where signals are highly variable, often bursty, and share frequencies in complex ways.

RTSAs, in contrast, capture the entire frequency band within their real-time bandwidth simultaneously and continuously. This ensures that no signal goes undetected — including brief pulses commonly associated with interference, disruptions, or burst transmissions. This gap-free signal capture is critical for fault diagnostics and frequency monitoring.

Every Hertz Counts

Real-time bandwidth — the range that the device can analyze simultaneously and without interruption — is a defining feature of modern RTSAs. In many applications, 40 or even 100 MHz of RTBW is no longer sufficient. In advanced 5G systems — especially in FR2 (*Frequency Range 2*, mmWave) — channel bandwidths can reach up to 400 MHz, with 100, 200, or 320 MHz being typical. Using an analyzer with only 160 MHz RTBW in such scenarios risks missing relevant portions of the signal.

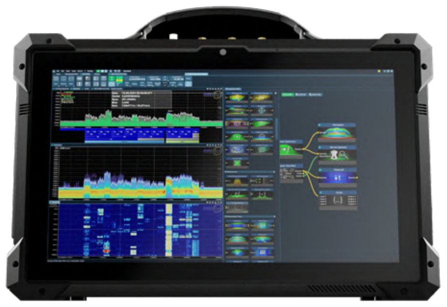


Figure 1: With the SPECTRAN® V6 Mobile, Aaronia AG has developed the world's first portable real-time spectrum analyzer with an RTBW of 490 MHz, a frequency range from 9 kHz to 140 GHz, and a sweep speed of 3 THz/s.

With an RTBW of at least 320 MHz, engineers can capture and analyze entire signals in a single sweep. This is especially vital in applications like carrier aggregation, where multiple frequency carriers are used simultaneously, or when evaluating spectral efficiency and interference. Without adequate bandwidth, blind spots may occur, potentially right where faults or interference lie hidden.

Mobile Powerhouse with 490 MHz RTBW

With the SPECTRAN® V6 Mobile, Aaronia AG has developed the world's first portable real-time spectrum analyzer, offering 490 MHz RTBW (**Figure 1**). This enables full analysis of even the 320 MHz-wide channels defined in the new IEEE 802.11ax standard. Covering a frequency range from 9 kHz to 140 GHz and featuring a sweep speed of 3 THz/s, Aaronia's analyzers are equipped for virtually any task. The 15″ display, offering up to 1,500-nit brightness, ensures readability even in daylight — ideal for outdoor applications.

With new PC boards, such as those integrated in the V6 Mobile series, Aaronia has addressed one of the major limitations of portable systems compared to lab-grade equipment: performance constraints. The boards, powered by the AMD® Ryzen 7 8845HS, are I/O powerhouses, enabling simultaneous operation of up to four SPECTRAN® V6 ECO analyzers (4× USB PD). Additionally, they feature SFP+ 10G, dual 2.5G Ethernet, 64 GB LPDDR5 RAM, and can be configured with up to 2 x 8TB SSDs. Expansion options include 2× M.2 2280 NVMe slots, 3× M.2 2242 NVMe slots, and a SIM card slot.

Figure 2: Equipped with analyzers from the SPECTRAN® V6 ECO series, Aaronia's portable systems are designed as fast sweepers for measurements in high frequency ranges and, at the same time, are tailored to the entry-level segment in terms of acquisition costs.

Figure 3: The SPECTRAN® V6 PLUS is the latest generation of high-performance real-time spectrum analyzers and sets new standards in the USB compact class with a sweep speed of over 1 THz/s.

Figure 4: The SPECTRAN® V6 5G also supports the WiGig 45 GHz (802.11aj) and 60 GHz (802.11ad/aj/ay) profiles, which are now included in the latest version of the RTSA Suite PRO for signal recording and data analysis.

Professional-Level Entry Segment

Equipped with SPECTRAN® V6 ECO analyzers (**Figure 2**), Aaronia's portable systems are designed for fast sweeps at high frequencies while being priced for the entry-level segment. Despite this, they offer features typically reserved for high-end instruments — such as a complex, fast-switching frontend filter bank or uninterrupted IQ data recording. For instance, the SPECTRAN® V6 ECO 100XA-6 covers a frequency range of 9 kHz to 6 GHz (optionally 7.25 GHz for Wi-Fi 6E) and delivers an RTBW of 44 MHz with a sweep speed of 500 GHz/s.

Especially at high frequencies in the 18 GHz range, environmental influences, atmospheric attenuation, and multipath effects present additional challenges that demand continuous monitoring and analysis. The SPECTRAN® V6 ECO models V6-100XA-18 and V6-150XA-18 offer coverage up to 18 GHz and a RTBW of more than 50 MHz. Their impressive standard sweep speed of 500 GHz/s can be increased via software license to over 3 THz/s — all over a single USB 3.1 port.

Ready for Wi-Fi 7

The introduction of 320 MHz channels in the IEEE 802.11be (Wi-Fi 7) standard presents new challenges in RF measurement. Full bandwidth capture of short-lived interferences, transients, or burst signals within the 320 MHz spectrum must be simultaneous and real-time. Spectrum dynamics at high frequencies further increase the complexity. Modern Wi-Fi signals use extremely short packet durations and adaptive modulations. This demands rapid FFT processing, large memory buffers for trigger data, and ideally, the ability to reconstruct signals with time-correlated I/Q data.

The SPECTRAN® V6 PLUS is the latest generation of high-performance real-time spectrum analyzers, setting new standards in the USB compact class with sweep speeds exceeding 1 THz/s (**Figure 3**). With a base RTBW of 245 MHz, they support gapless IQ data streaming — unmatched at this bandwidth. For demanding applications, 490 MHz RTBW is available, ideal for analyzing 320 MHz channels. This enables the detection of intermittent or short-duration signals that traditional methods would likely miss. With an ultra-low DANL of -170 dBm/Hz, SPECTRAN® V6 Mobile systems are the perfect tools for precise RF measurements.

Built for High Frequency Bands

Fast spectrum updates and transient signal detection are especially important for analyzing frequency-hopping signals or locating intermittent interference.

5G utilizes a wide range of frequency bands, many of which are significantly higher than those used in previous cellular technologies, such as the extremely high frequency (EHF) band from 30 GHz to 300 GHz. Analyzing such signals requires spectrum analyzers with high resolution and sufficient bandwidth to reflect their complex structure. Moreover, 5G networks employ beamforming and Massive MIMO technologies, further complicating signal analysis.

For these needs and more, Aaronia offers the SPECTRAN® V6 5G variants (**Figure 4**). These devices also support WiGig 45 GHz (802.11aj) and 60 GHz (802.11ad/aj/ay) profiles, now fully integrated into the latest version of the RTSA Suite PRO for signal capture and data analysis.

Reliable IQ Data Analysis

All SPECTRAN® V6 Mobile variants feature the unique ability to stream wideband IQ data continuously and without loss to a PC for recording and later analysis via the RTSA Suite PRO. IQ data analysis is increasingly critical, but the challenge lies in managing the massive data volumes generated — often bloated with unnecessary content, which hampers the search for relevant events.

Aaronia addresses this with an innovative solution that allows selective recording of predefined events. For instance, the system can be configured to capture only activity around 2.4 GHz related to Wi-Fi, ignoring everything else. Multiple events can be defined simultaneously, allowing technicians to build a compact database of all relevant events while using only a fraction of the storage capacity. This also makes 24/7 recording more feasible with minimal disk space. Meaningful IQ data logging and post-event analysis — widely considered the pinnacle of RF diagnostics — is an area where Aaronia excels.

www.aaronia.com

250609-0

Q&A With Anastasiia Nosova

Founder of the popular tech podcast, "Anastasi in Tech"

The Future of AI Is Forged in Silicon

An Interview with Anastasiia Nosova

About Anastasiia Nosova

Anastasiia Nosova is a leading expert in computing hardware for AI. She holds an MSc degree in Electronics and Computer Science and spent nearly a decade in R&D at Infineon Technologies, where she designed cutting-edge chips for autonomous driving. Leveraging her expertise in microchip R&D, Ms. Nosova founded a highly influential YouTube podcast "Anastasi in Tech" [1] on semiconductors and AI, building a community of over 300,000 followers across social media. Her work is highly recognized by industry giants like AMD, NVIDIA, Google, and leading research institutions, including Stanford, UC Santa Barbara, and IMEC.

Questions by the Elektor Team

As AI workloads push the limits of computing, graphics card chips have taken center stage. In this interview, Elektor speaks with Anastasiia Nosova — founder of the popular tech podcast *Anastasi in Tech* — about the future of AI hardware, GPUs, and the innovations reshaping the semiconductor industry.

Elektor: Let's start with the fundamentals: What is a GPU, and how did NVIDIA — once focused on graphics for gaming — evolve into a $4 trillion leader at the forefront of the AI revolution?

Anastasiia Nosova: A GPU's architecture is key. Unlike a CPU that handles a few complex tasks sequentially, a GPU uses thousands of simple cores to perform many calculations in parallel. This design, originally for graphics, is perfectly suited for the massive matrix multiplications at the heart of AI workloads. NVIDIA's success stems not only from delivering the right hardware at the right moment, but also from their ability to identify key market trends early, and go all in. Most critically, they made a bold, strategic investment in the CUDA software platform. CUDA became the standard for programming GPUs for AI and scientific tasks, creating a powerful software "moat" that gave them an incredible head start.

Elektor: That's a powerful first-mover advantage. But in a market this large, is there serious competition? Why aren't we seeing other $4-trillion chipmakers?

Anastasiia: The competition is significant and growing. Traditional rivals like AMD are offering powerful alternatives, its MI325X GPUs show strong performance and better cost-efficiency than NVIDIA's H100 and H200 in specific large-model inference tasks. A particularly interesting trend is that the world's largest companies by market cap — Google, Microsoft, Amazon, and Apple — are all designing their own chips, making it a key differentiator — and I break it all down on my podcast. Google alone shipped $6 billion worth of TPUs internally in 2024. Meanwhile, innovative startups like Cerebras are pushing the envelope with radical wafer-scale designs. While NVIDIA maintains a clear lead today, it is being challenged on all fronts and must continue to innovate at a rapid pace.

Elektor: With all these players building chips, are we going to see an oversupply of AI hardware anytime soon?

Anastasiia: An oversupply is highly unlikely. The demand for AI compute is exploding due to widespread adoption and, more importantly, the growing complexity of AI models. For instance, advanced reasoning models require up to 10 times more compute than traditional ones. Despite optimization efforts, the overall trend is clear: compute is estimated to grow 4× to 5× annually, far outpacing the current supply.

Elektor: Are we keeping up with this staggering demand?

Anastasiia: Hardware is struggling to keep up with the pace of AI demand. Research from EpochAI shows that hardware performance is improving by only 1.3× per year, while demand is growing 4× to 5× annually. Building more data centers isn't a sustainable solution in the long run. These new "AI gigafactories" already consume massive amounts of power, 500 megawatts and rising. At the same time, networking and compute bottlenecks are becoming increasingly severe. It's clear that incremental improvements won't be enough. We need fundamental innovations in computing.

Elektor: You mentioned networking. Why is it becoming so critical?

Anastasiia: In AI data centers, thousands of GPUs must work together as one large, distributed system, constantly exchanging huge amounts of data with minimal delay. Any latency or bandwidth bottleneck slows down the entire pipeline, no matter how powerful each GPU is. Traditional electrical interconnects are becoming a severe bottleneck, hitting physical limits on speed and power consumption. The clear solution is optics. Light-based communication is now moving from connecting servers (“scale-out”) to connecting chips within a single system (“scale-up”). This shift, with technologies like co-packaged optics, is key to unlocking the next level of performance.

Pushmeet Kohli of Google DeepMind about their most advanced AI agent for science [2].

An even bigger challenge in scaling AI data centers is power, not just the amount, but how efficiently it’s used. These facilities demand enormous and continuous electricity, often exceeding what local power grids can supply. I’ve been talking about this for a while on my YouTube channel [1]. Whoever secures access to cheap, abundant power will ultimately win the AI race.

Elektor: It seems every so often we hear news that “Moore’s Law is dead,” but then a new, smaller “nanometer technology” is announced. Can you comment on this cycle?

Anastasiia: Moore’s Law isn’t dead, but it has certainly evolved. Around the 22 nm node, the industry moved from 2D “planar” transistors to 3D FinFET structures to overcome physical limits. Now, next-gen AMD and Apple chips are adopting “nanosheet” transistors, marking the next milestone in transistor scaling. Today, progress is less about shrinking and more about smart 3D designs that boost performance and density. In the next decade, however, the real leaps in performance may come from new materials, such as 2D and carbon-based materials. Eventually, this all leads to the idea of CMOS 2.0, a new way of building chips where each layer is optimized for a specific task. Instead of cramming everything into a single flat chip (as we’ve done for decades), we can use the best material and technology for each layer, whether it’s for AI, graphics, or something else.

Anastasiia at the “Green Computing in the AI Era,” where she discussed energy-efficient AI, highlighting advances in photonics, quantum, and probabilistic computing.

Elektor: Looking ahead, which specific technology do you see as the most transformative for the future of AI computing?

Anastasiia: The future won’t be defined by a single silver bullet, but by a combination of technologies. To move forward, we need to advance all three pillars of AI hardware together: compute, memory, and interconnect. Improving just one creates new bottlenecks in the others. Personally, I’m most excited about the potential of silicon photonics, not just for connecting chips, but for computing itself. Doing calculations with light offers major gains in speed and energy efficiency. I believe the next big leap in AI hardware will come from tightly integrating photonics with electronics.

Elektor: For engineers and startups looking to enter the AI hardware space today, where are the biggest opportunities still open?

Anastasiia: This is a very interesting question. In my opinion, the most fertile ground is at the intersection of electronics and photonics. I am personally very interested in optical computing; there are still immense performance and efficiency gains to be explored in that domain. Another area, which is often overlooked but absolutely critical, is power conversion and delivery. The challenge of efficiently converting an 800 V DC supply down to the 0.7 V needed by the processor core is immense. Furthermore, advanced technologies like backside power delivery networks will flip traditional chip power design upside down, creating a huge moment for innovation. Finally, the developing chiplet ecosystem will open new doors, allowing startups to design specialized components and compete in the hardware race without the prohibitive cost of designing a full, monolithic system-on-chip.

Elektor: Is there a risk that the AI compute race becomes too centralized, with only a few hyperscalers or labs controlling access to the most powerful systems?

Anastasiia: Absolutely, that’s already happening. However, I would be more concerned about the concentration of compute in the hands of a few leading AI labs rather than just the hyperscalers who provide the infrastructure. The open-source strategies we see from some labs are a very positive and viable way to democratize access to powerful AI. But we have to be realistic. If there is a rapid, discontinuous advance in AI performance from one of the closed-source labs, a centralization of “power” is somehow inevitable. From a European perspective, in my opinion, we should be treating this as a matter of strategic autonomy and definitely invest more capital in chip development, data centers and AI capabilities.

250589-01

WEB LINKS

[1] “Anastasi in Tech” on YouTube: https://www.youtube.com/c/AnastasiInTech
[2] “AI Has Never Been Able To Do It - Until Now (ft. Deepmind)” on YouTube: https://www.youtube.com/watch?v=kucIgsS6wrw

Autonomous Sensor Node v2.0 (System Architecture)

Solar-Powered Sensing Platform with Integrated GPS, LoRaWAN, and More

By Saad Imtiaz (Elektor)

The second version of the Sensor Node is a complete redesign integrating an ESP32-S3 MCU, a RAK3172 LoRaWAN module, GPS, MPPT battery charging, and timer control — all onto a single PCB. Built for remote deployment, the system focuses on efficient power usage, local data logging, and field flexibility. This article presents the system architecture, hardware choices, and design considerations that form the foundation of this next-generation, low-power sensing platform.

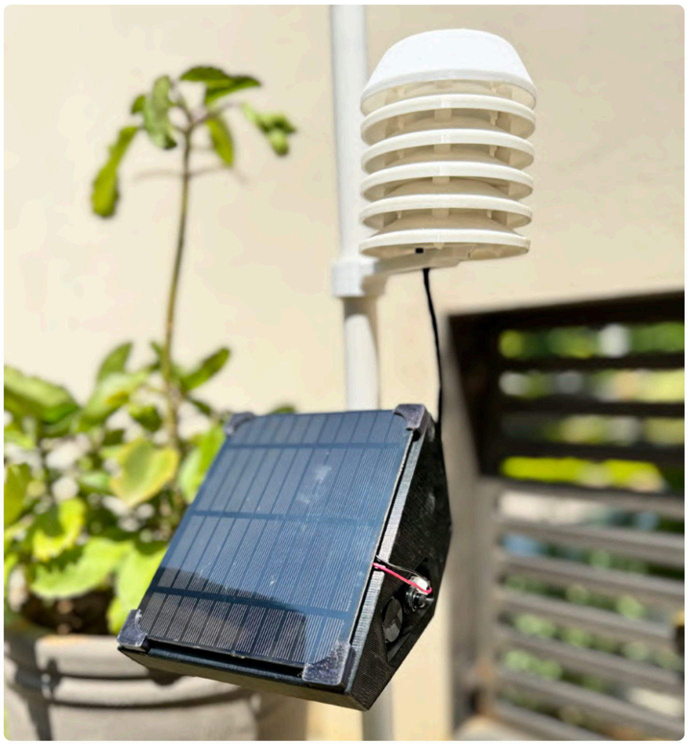


Figure 1: The Solar Powered LoRaWAN Sensor Node v1.0.

After testing and validating the first version of a sensor node built using modular components (**Figure 1**) [1], the need for a more integrated, compact, and power-optimized design became apparent. The earlier design, while functional, relied on multiple breakout boards, which added wiring complexity, size, and power losses. The new version is designed from scratch to address these issues by integrating all required modules into a single board with careful consideration for battery life, expandability, and use in solar-powered remote environments.

With this version, the target outdoor deployments where the node must operate autonomously, often for months, on limited solar input. At the same time, the node is designed with flexibility in mind, not just for low-power mode but also for use as a gateway or local edge device where power is available. An SD card is also included for local data logging, making the node resilient to communication dropouts or for use in offline-only modes.

System Architecture Overview

The system architecture for the second version of the Sensor Node is designed around modular power control and efficient communication domains, all orchestrated by the ESP32-S3 microcontroller from Espressif. The design targets solar-powered, autonomous deployments, where minimizing idle power consumption is critical.

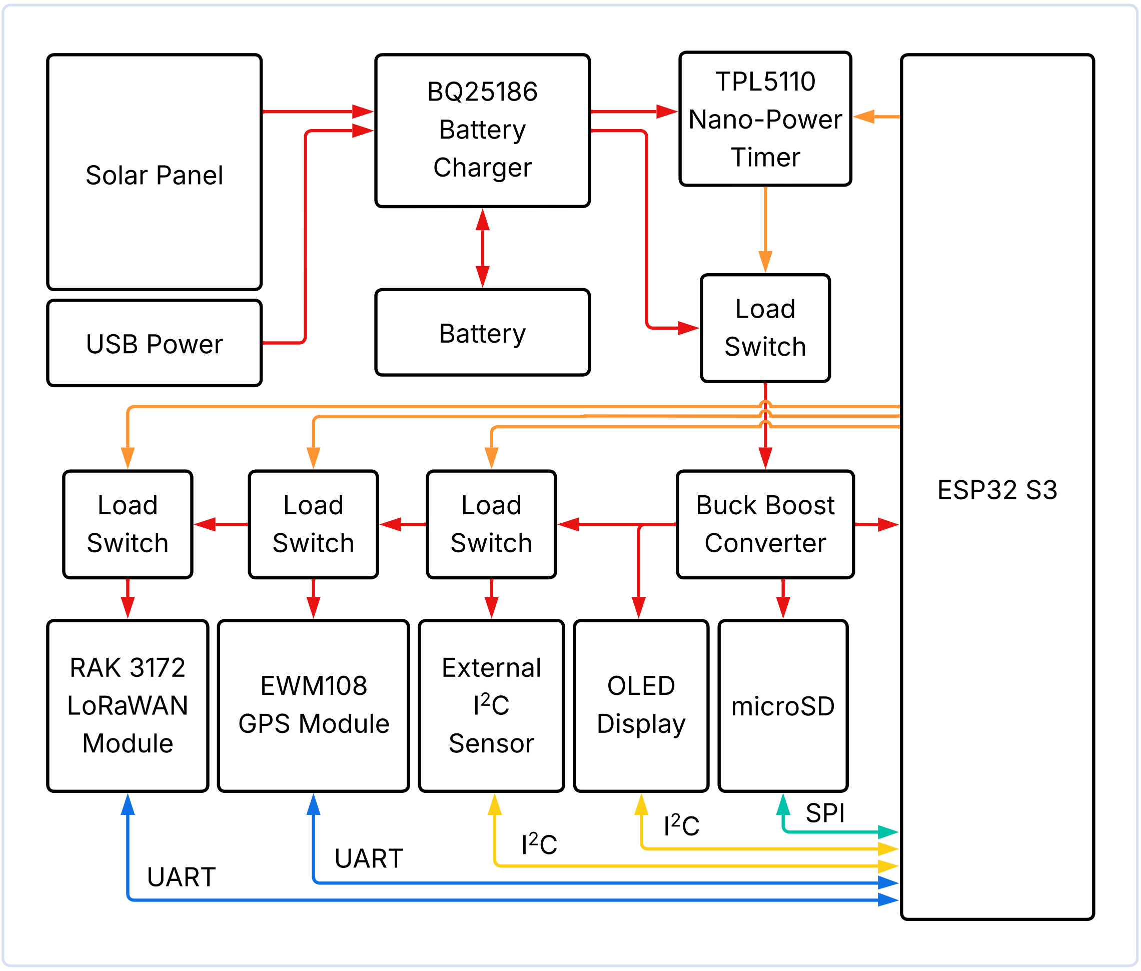


Figure 2: The system block diagram shows power flow, control signals, and module interconnections.

The BQ25186 battery charger from Texas Instruments offers a wide operational V_{IN} range for high impedance sources such as solar cells. Hence, the power can be sourced either from a solar panel or via an USB input; the BQ25186 manages charging of a single-cell Li-Ion battery. From there, power flows through a load switch controlled by the TPL5110 nano-power timer from TI, which serves as the system's primary power controller. The TPL5110 physically enables or disables power to the entire circuit at fixed intervals, as defined by an external resistor.

Once enabled, power is routed to the TPS631000D buck-boost converter from TI, which regulates a stable 3.3 V output from the variable battery voltage. This ensures reliable operation of the ESP32-S3 and all peripherals, regardless of the battery's charge state. This combination of hardware-based shutdown and efficient regulation brings the system's idle current consumption down to the nanoamp range.

Once powered up, the ESP32-S3 initializes and begins a structured sequence: activating required peripherals through TPS22917 load switches from TI, acquiring GNSS data from the Ebyte EWM108-GN05 GPS module, collecting sensor data, storing it to the microSD card, and finally transmitting the data using the RAK3172 LoRaWAN module from RAKwireless. The microcontroller communicates with the GPS and LoRaWAN modules via UART, while I²C is used for sensors and the OLED display, and SPI handles the SD card interface. Each of these interfaces is only active during the system's brief operational window to minimize energy use.

All peripherals draw power through dedicated TPS22917 switches, which are controlled by GPIO lines from the ESP32-S3. These lines, highlighted in orange in the system block diagram (**Figure 2**), not only toggle individual modules but also signal completion of the system's task cycle. Once data has been logged and transmitted, the ESP32-S3 sends a High pulse to the *DONE* pin on the TPL5110, instructing it to cut off power to the system, effectively resetting the timer until the next wake cycle.

A dedicated microSD card provides local data storage, enhancing resilience in case of LoRaWAN transmission failures or low connectivity environments. It also enables long-term high-frequency logging for offline analysis or AI model training, making the node suitable not just for real-time alerts but also for data-driven insights.

Together, this architecture allows for a clean flow of operation from power-up, controlled data acquisition and transmission, to total shutdown. The design enables the system to operate independently for months in field conditions using only solar energy, with tight control over every milliamp consumed. This kind of block-level modular control is what makes the V2 node both energy-efficient and versatile for different deployment conditions.

Hardware Selection

Let's consider the pieces of hardware incorporated in this project.

ESP32-S3 Microcontroller

The ESP32-S3 is the central controller for the node. It offers a dual-core Xtensa LX7 processor with ultra-low-power operation modes, capable of reaching sub-10 µA deep sleep currents. It is also equipped with hardware acceleration for AI workloads, which allows this node to be repurposed for Edge ML tasks when operating as a gateway or hub. In this project, the ESP32-S3 handles all logic, including power sequencing, data logging, UART communication with the radio and GPS, and system diagnostics. Its flexible I/O and low-power consumption make it suitable for both battery-operated and mains-powered scenarios.

RAK3172 LoRaWAN Module

The RAK3172 module is based on the STM32WLE5 SoC and is responsible for LoRaWAN communication. It is integrated using UART with AT command support, reducing firmware complexity. The module supports global LoRa frequency plans and is configurable for Class A/B/C operation. During data transmission, it draws around 87 mA at +20 dBm output power. When not in use, it can be fully powered off through a load switch or set to a sleep mode where it consumes around 1.7 µA, aligning with the project's tight energy budget. Its compact form factor and RF performance make it an ideal fit for long-range, low-duty-cycle communication.

EWM108-GN05 GPS Module

The GPS module used is the EWM108-GN05 from Ebyte, which supports GPS, GLONASS, Galileo, BDS, and SBAS constellations. It offers sub-meter accuracy, high sensitivity (cold start -148 dBm, tracking -162 dBm), and a fast time-to-fix. During active acquisition and tracking, the module consumes around 47.3 to 47.8 mA. However, in this design, it is powered only during transmission windows through a TPS22917 switch, bringing its sleep current down to just 14 µA. This allows periodic location tagging without a continuous energy penalty, particularly useful for mobile or semi-fixed deployments.

BQ25186 Battery Charger

The BQ25186 is a linear charger with integrated MPPT for solar energy harvesting. It supports input voltages from 3 V to 18 V and charging currents up to 1 A. Most importantly, it features a shutdown current of 15 nA and a quiescent current of just 4 µA when operating from the battery — critical for maintaining long-term performance with limited solar availability. It includes programmable charging profiles, termination control, and system power path management, allowing simultaneous charging and system operation without discharging the battery.

TPL5110 Nano-Power Timer

The TPL5110 is a nano-power timer IC that physically disconnects the system from the battery when not in use. It operates with just 35 nA

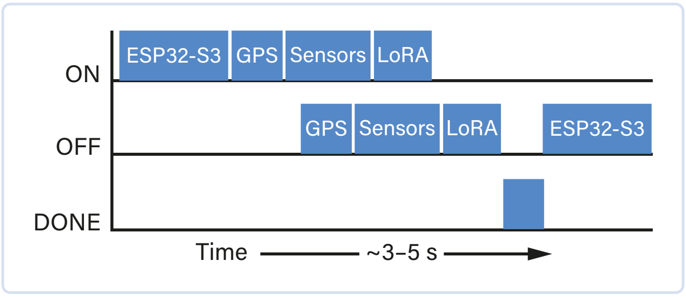


Figure 3: System Operation Timeline: wake cycle timing of GPS, sensors, LoRa transmission, and system shutdown.

current draw and wakes the system at fixed intervals (e.g., 15 or 30 minutes). Once awake, the ESP32-S3 performs all tasks and sends a *done* signal to shut the power again. This architecture bypasses the limitations of deep sleep modes by eliminating idle leakage currents from both the MCU and peripheral components, significantly extending battery runtime in low-activity use cases.

TPS22917 Load Switches

Multiple TPS22917 load switches are used to gate power to modules like the GPS and LoRa radio. These switches are controlled directly from the ESP32-S3, feature low on-resistance (~80 mΩ), and consume only 10 nA in the off state. This allows precise control over when peripherals receive power, enabling aggressive power savings without needing to reconfigure voltage rails or regulator enable lines. It also simplifies the schematic and power routing on the PCB.

TPS631000D Buck-Boost Converter

This converter maintains a stable 3.3 V system rail across the full battery voltage range. With support for input voltages from 1.6 V to 5.5 V and up to 1.5 A continuous current output, the TPS631000D ensures all components operate reliably regardless of battery condition. The quiescent current is just 8 µA, and it uses a fixed 2 MHz switching frequency for compact external components and low output ripple. This makes it well-suited for power-constrained wireless designs like this one.

System Operation Timing

The system operation timeline (**Figure 3**) illustrates how the node will wake up, perform its tasks, and return to a low-power state. After the TPL5110 powers the system, the ESP32-S3 turns on the GPS, sensors, and LoRa module in sequence. Each is active only as long as needed — typically 3 to 5 s total — before being shut down. Once data is logged and transmitted, the ESP32-S3 sends a *Done* signal to the TPL5110, cutting power to the entire system. This efficient timing will potentially keep standby current below 5 µA, enabling long-term operation from a small solar-powered battery.

Current Development Status and Next Steps

As of now, the entire system is being integrated into a single multi-layer PCB, which is in the layout and routing phase. The focus is on minimizing power losses, optimizing the ground planes for clean analog signals, and isolating high-frequency circuits like the switching converters and GPS module. Once the PCB is ready, the next step will be hardware

Table 1: Power Consumption Per Module.

Module	Active	Sleep
ESP32-S3	20–80 mA	~5 µA (Deep Sleep)
RAK3172 (LoRa)	~87 mA (TX)	~1.7 µA
EWM108-GN05 (GPS)	~47.5 mA	~14 µA (Standby)
BQ25186 (Charger)	Load dependent	4 µA / 15 nA (Shutdown)
TPL5110 (Timer)	—	35 nA
TPS22917 (Switch)	<1 µA (On)	~10 nA (Off)
TPS631000D (DC/DC)	Load dependent	~8 µA (Quiescent)
microSD Card	20–30 mA	~100–250 µA (Standby)
OLED Display	10–20 mA	Few µA or powered off

bring-up — testing each subsystem individually, including validating the charging efficiency, GPS acquisition time, and LoRaWAN transmission range. Software development will start in parallel, beginning with a simple state machine to control the power-on sequence, gather sensor data, and transmit it over LoRaWAN.

The firmware will be designed to be lightweight and modular, using the Arduino framework. A simple UART-based interface will be used to control the RAK3172 LoRaWAN module, while GPS data will be parsed over UART as well. I^{2}C-based environmental sensors such as temperature, humidity, and air quality modules will also be integrated, similar to the earlier project. Power consumption of each module will be carefully considered during development, as outlined in **Table 1**, to ensure efficient energy use in both active and sleep states.

Where This Project Is Heading

Once completed, this project aims to deliver a compact, solar-powered, long-range sensor node ready for harsh outdoor environments. By reducing the number of external modules and optimizing every power path, this v2.0 design significantly improves over the first prototype. The use case remains the same — remote monitoring in places where running a power cable is impractical, and where environmental data needs to be collected and transmitted over long distances using LoRaWAN.

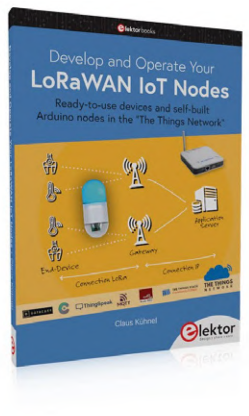
250545-01

Questions or Comments?

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

About the Author

Saad Imtiaz, Senior Engineer at Elektor, is a mechatronics engineer who has extensive experience in embedded systems and product development. His journey has seen him collaborate with a diverse array of companies, from innovative startups to established global enterprises, driving forward-thinking prototyping and development projects. With a rich background that includes a stint in the aviation industry and leadership of a technology startup, Saad brings a unique blend of technical expertise and entrepreneurial spirit to his role at Elektor. Here, he contributes to project development in both software and hardware.



WEB LINK

[1] Saad Imtiaz, "An Autonomous Sensor Node," Elektor 9-10/2024: https://elektormagazine.com/240354-01

Precise Positioning

Bluetooth Channel Sounding Tested

By Tam Hanna (Hungary)

Bluetooth beacons have always inspired marketing specialists and developers, using it to position objects that cannot use GPS due to a lack of line of sight. In recent years, the Bluetooth SIG has introduced novel indoor positioning methods, which we will look at in this article. One of the newer technologies is channel sounding, which we will try out in practice.

The use of standards-driven implementations is preferable to self-built or proprietary systems, if only because the — certainly not beyond all criticism — Bluetooth SIG guarantees a certain consistency in implementations due to its market power.

The experiments discussed here are based on Silicon Labs technology. However, while you are reading this article, other manufacturers are likely already delivering their microcontrollers compatible with these methods. Also, it is expected that smartphone manufacturers will soon "follow suit," so that over the next few years a comprehensive positioning ecosystem will develop based on the technologies discussed in this article.

Indoor Positioning

Given the long history of Bluetooth technology, various standards and methods for positioning have been developed over the years. The easiest approach is certainly to use the RSSI — signal strength allows a basic estimate of distance, but not the direction between the two participating parties. The next step is *Direction Finding* — these are methods to determine directional information between two devices.

It is important to note that the Bluetooth SIG provides two methods here: in the case of *Angle of Arrival* (AoA), fixed devices known as *Locators* determine the position, while in the case of *Angle of Departure* (AoD), the "mobile" device determines its position relative to the fixed stations now referred to as *Beacons*.

For AoA, the asset sends a signal that is received by the locators via an antenna array. The collected IQ information then enables determination of the distance between the two systems.

For AoD, the beacons send a signal using multiple antennas — since the same frequency must be used, the antennas are activated in sequence (keyword *Time Division*). The asset can then recognize differences in the IQ signals. **Figure 1** and **Figure 2** present a comparison of the methods.

The next step is converting directional information to full location data. The answer is triangulation: deploying multiple base stations allows complete positioning.

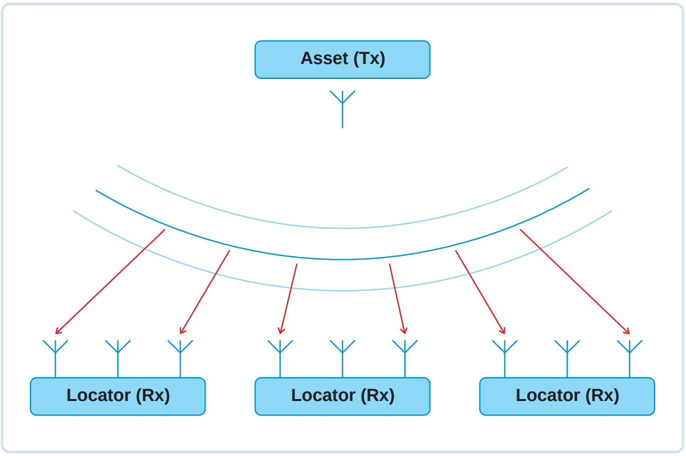


Figure 1: Angle of Arrival (Source: Silicon Labs).

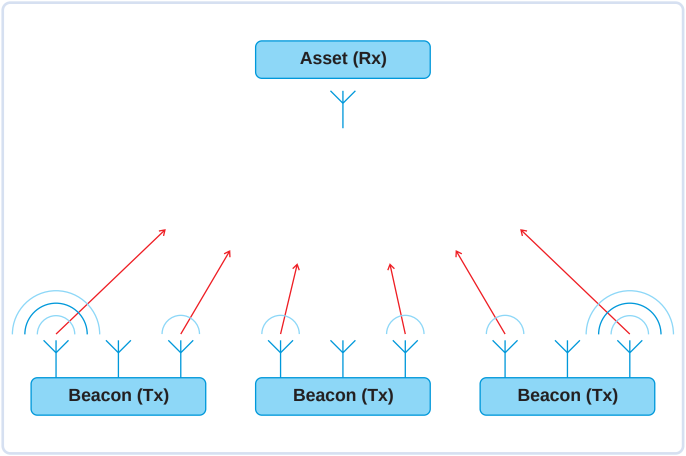


Figure 2: Angle of Departure (Source: Silicon Labs).

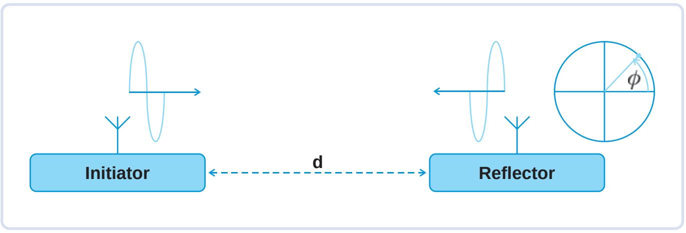


Figure 3: Phase shift is the base technology behind Channel Sounding (Source: Silicon Labs).

Figure 4: These boards can be used for localization.

Bluetooth Channel Sounding — the latest variant of the technology — instead works, as shown in **Figure 3** with phase shift.

A single channel allows the determination of an approximate distance — whether the device is d, 2 × d or N × d distance units away cannot be reliably determined using only a single channel. The measurement is completed by using all 72 channels available in the standard — several different values of d allow determination of the distance. A precision of ±20 cm is reported to be possible.

Excellent Documentation Available

Silicon Labs provides extensive documentation on the various methods at URL [3], which also covers the mathematical basics and is truly a *must-read* for anyone interested in amateur radio, radar, and similar topics.

Setting Up the Development Environment

Now we want to perform our first practical experiments. For this, we use two EFR32xG24 Channel Sounding Dev Kit boards from Silicon Labs (**Figure 4**). One will serve as the *Reflector*, the other as *Initiator*.

It should be noted that Silicon Labs also offers other boards based on the EFR32xG24 SOC. However, using the Channel Sounding function requires a 40 MHz crystal, which not all boards provide.

Why 40 MHz?

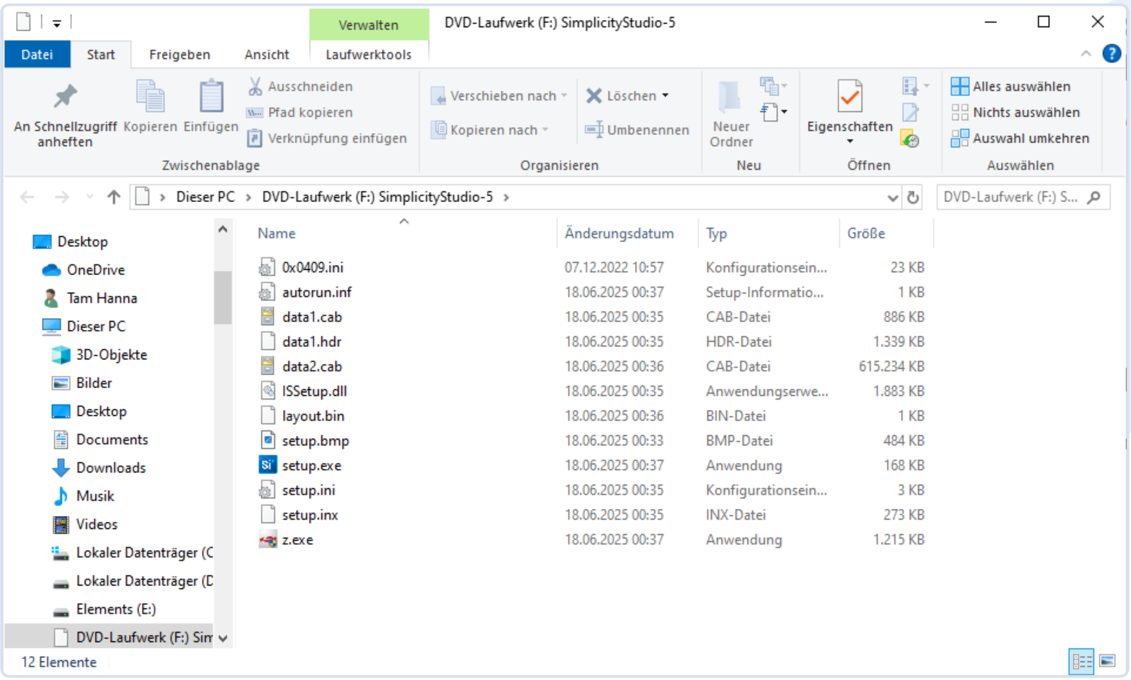
The use of channel sounding on Silicon Labs hardware requires the RTT (Round Trip Time) function. It is only available on board versions with a 40-MHz crystal and is only tested on these. Using other operating speeds would require a complete reconfiguration of the hardware, which is not supported by Silicon Labs and is therefore not recommended.

To get started with the Silicon Labs development environment called Simplicity Studio, which is based on Eclipse CDT, you need an account on the manufacturer's website. Visit URL [1], where downloads are available for Windows, Mac OS, and Linux. If, like the author, you want to work with Windows, select the *Windows Installer* link. On the login screen that then appears, you can create a new account to start the download process.

After successfully completing the registration process, we can call the URL again and click the download link. Interestingly, Silicon Labs delivers the software as an .iso file — at the time of writing, it was called *SimplicityStudio-5.iso*.

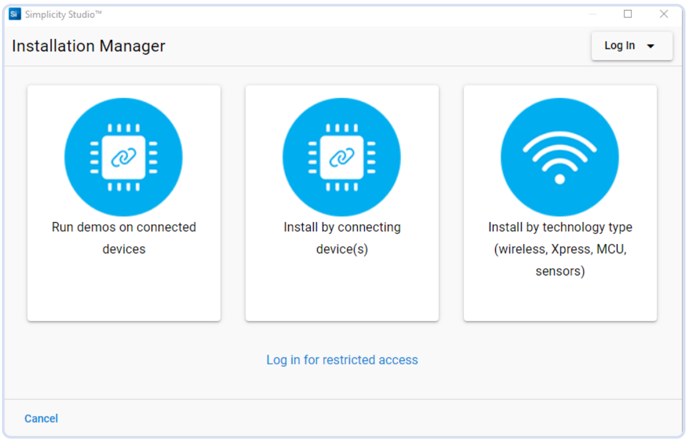
Double-clicking in Windows 10 opens this file — as shown in **Figure 5** — just like a classic archive.

The most convenient way to install is to extract the entire contents



Name	Änderungsdatum	Typ	Größe
0x0409.ini	07.12.2022 10:57	Konfigurationsein...	23 KB
autorun.inf	18.06.2025 00:37	Setup-Informatio...	1 KB
data1.cab	18.06.2025 00:35	CAB-Datei	886 KB
data1.hdr	18.06.2025 00:35	HDR-Datei	1.339 KB
data2.cab	18.06.2025 00:36	CAB-Datei	615.234 KB
ISSetup.dll	18.06.2025 00:35	Anwendungserwe...	1.883 KB
layout.bin	18.06.2025 00:36	BIN-Datei	1 KB
setup.bmp	18.06.2025 00:33	BMP-Datei	484 KB
setup.exe	18.06.2025 00:37	Anwendung	168 KB
setup.ini	18.06.2025 00:35	Konfigurationsein...	3 KB
setup.inx	18.06.2025 00:35	INX-Datei	273 KB
z.exe	18.06.2025 00:37	Anwendung	1.215 KB

Figure 5: This is the first step.


Simplicity Studio™

Installation Manager

Log In

Run demos on connected devices

Install by connecting device(s)

Install by technology type (wireless, Xpress, MCU, sensors)

Log in for restricted access

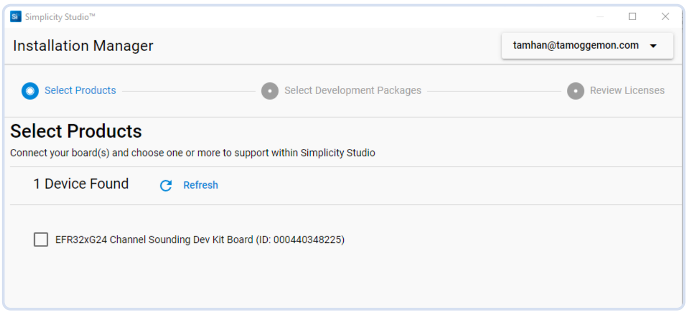
Cancel

Figure 6: The Silicon Labs IDE supports login with a Silicon Labs account.

of the .iso file into a temporary directory. Then run the *setup.exe* file, which starts a setup process reminiscent of any typical Windows application. Among other things, you must accept a license agreement. Installation prompts also appear, allowing the IDE to deploy various drivers for Segger command devices — these must be accepted because communication with Silicon Labs hardware generally uses Segger. After installation, the *Installation Manager* appears as shown in **Figure 6**, where you should first click *Log in for Restricted Access*.

After logging in, it is advisable to wait about 5 minutes before closing the IDE. During the processing of the *Progress Information* dialog, which freezes from time to time, Windows permission prompts will appear. These must be accepted, as they enable additional hardware drivers to be installed.

In addition to the integrated development environment, an SDK called Simplicity SDK is required, with source code available for download on GitHub [2]. A convenient way to deploy it is to use the dialog shown in Figure 6 and select *Install by Connecting Device*.


Simplicity Studio™

Installation Manager

tamhan@tamoggemon.com

Select Products — Select Development Packages — Review Licenses

Select Products

Connect your board(s) and choose one or more to support within Simplicity Studio

1 Device Found   Refresh

☐ EFR32xG24 Channel Sounding Dev Kit Board (ID: 000440348225)

Figure 7: Simplicity Studio recognizes the test board.

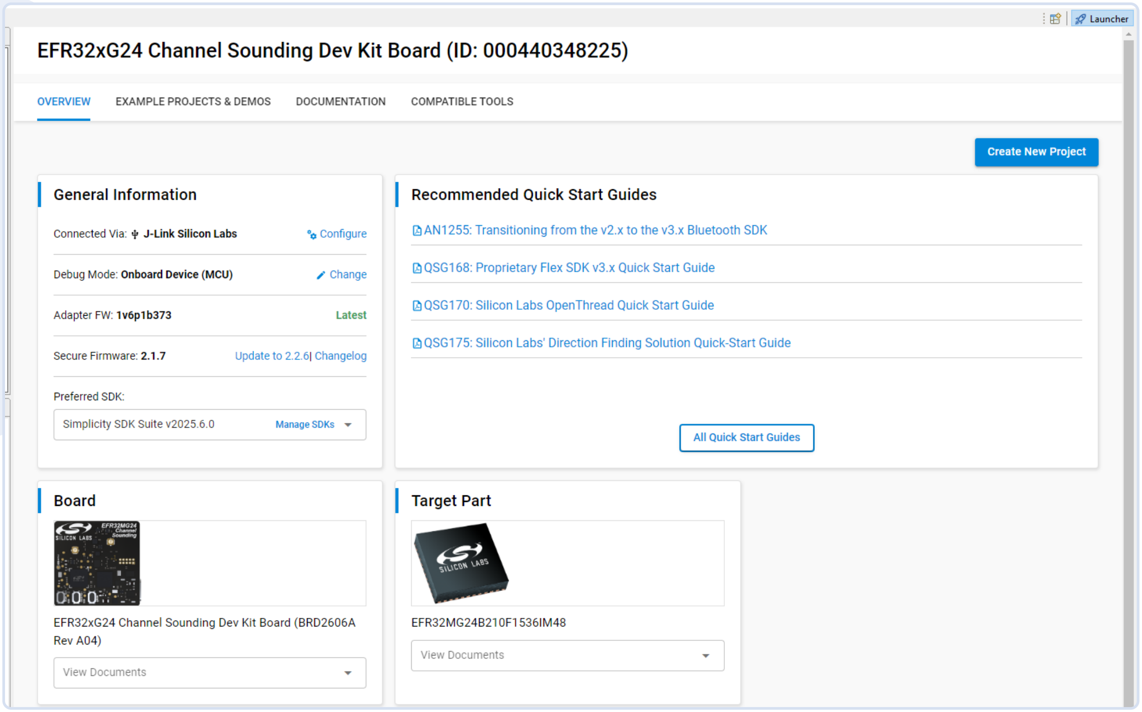
Simplicity Studio can automatically detect connected evaluation boards and recognize them as targets for the project.

After the installation manager dialog appears, one of the two evaluation boards is connected to the workstation using a USB-C cable. Wait until it is recognized as *EFR32XG24 Channel Sounding Dev Kit Board* — Simplicity Studio will initially identify the board as Segger J-Link and only recognize the attached microcontroller after a few seconds (**Figure 7**).

Select the checkbox in front of the found device and click *Next* to prompt Simplicity Studio to update the software list. Then choose *Auto* to download all recommended software packages. Confirming triggers the automatic deployment of the required code elements. An Internet connection is required for this, as the system must download several hundred MB.

After downloading, it is necessary to restart Simplicity Studio to update the various catalogs included in the product.

If the evaluation board remains connected to the workstation during the IDE restart, it will be listed as connected on the start screen. Then click the *Start* button to load the board-specific project generator shown in **Figure 8**.


Launcher

EFR32xG24 Channel Sounding Dev Kit Board (ID: 000440348225)

OVERVIEW   EXAMPLE PROJECTS & DEMOS   DOCUMENTATION   COMPATIBLE TOOLS

Create New Project

General Information

Connected Via: J-Link Silicon Labs   Configure

Debug Mode: Onboard Device (MCU)   Change

Adapter FW: 1v6p1b373   Latest

Secure Firmware: 2.1.7   Update to 2.2.6 | Changelog

Preferred SDK:

Simplicity SDK Suite v2025.6.0   Manage SDKs

Recommended Quick Start Guides

AN1255: Transitioning from the v2.x to the v3.x Bluetooth SDK

QSG168: Proprietary Flex SDK v3.x Quick Start Guide

QSG170: Silicon Labs OpenThread Quick Start Guide

QSG175: Silicon Labs' Direction Finding Solution Quick-Start Guide

All Quick Start Guides

Board

EFR32xG24 Channel Sounding Dev Kit Board (BRD2606A Rev A04)

View Documents

Target Part

EFR32MG24B210F1536IM48

View Documents

Figure 8: The options shown here are optimized for the connected evaluation board.

In the next step, switch to *Example Projects and Demos*, where Simplicity Studio offers a list of examples compatible with the connected development board.

The project skeletons intended for experiments with Channel Sounding contain the string *SoC CS* in the name — specifically, there is the Initiator and the Reflector. The Reflector acts as a base station, allowing the Initiator to measure the distance between itself and this base.

A convenience feature of the environment: For many sample projects, there is also a tile labeled *Demo*. Clicking *Run* here can execute the project on the connected development board. In the following steps, however, we will use the *Create* option to load a complete and analyzable code skeleton into the IDE.

Starting the Reflector

Now we want to start the reflector. For this, click the *Create* button in the *Bluetooth - SoC CS Reflector* tile to start the project generator. For now, we want to keep the suggested project name and only check if the path entered in the *Location* field is reasonable and contains no special characters or other potential issues.

The *With Project Files* option is interesting. As with other IDEs, Silicon Labs also allows you to include the SDK and Co. as part of the project solution — for the following steps, we'll stick with the *Standard* option, which keeps most of the SDK outside the project to reduce project size.

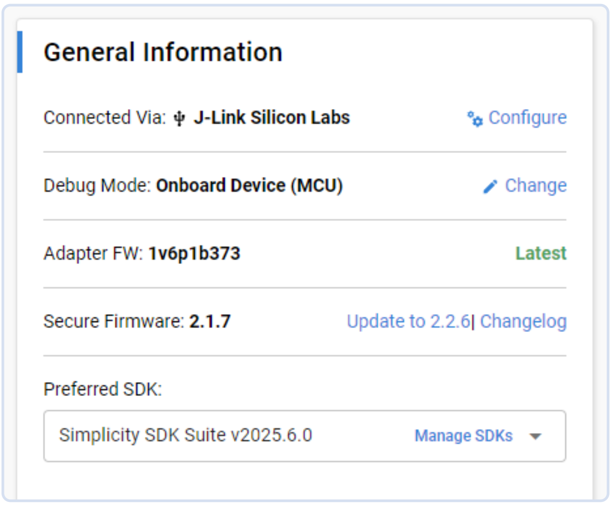
After clicking *Finish*, Simplicity Studio creates a project skeleton — this process generally follows what you may know from other Eclipse-based IDEs.

As is typical with Silicon Labs, the *main.c* file provides only a minimal work loop. The most important file is *App.c*. The file *config/cs_reflector_config.h* contains constants that configure the behavior of the Bluetooth stack. One example is the power output over the antennas.

It should also be noted that this sample — any similarities to Samsung's Bada are purely coincidental — mainly wires up pre-built components of the wireless stack. Particularly important is the folder *simplicity_sdk_2025.6.0 → app → bluetooth → common*, which provides some of the components discussed in the following steps.

Generally, the Bluetooth stack works asynchronously — the method `sl_bt_on_event(sl_bt_msg_t *evt)` serves as entry point and is responsible for configuring radio parameters.

The method `ble_peer_manager_on_event_reflector` is also interesting. It is shown here without various logging calls and enables the handling of connections:


General Information

Connected Via: J-Link Silicon Labs — Configure

Debug Mode: **Onboard Device (MCU)** — Change

Adapter FW: **1v6p1b373** — Latest

Secure Firmware: **2.1.7** — Update to 2.2.6 | Changelog

Preferred SDK:

Simplicity SDK Suite v2025.6.0 — Manage SDKs

Figure 9: The module base firmware is by definition outdated upon delivery.

```
void ble_peer_manager_on_event_reflector
     (const ble_peer_manager_evt_type_t *event) {
  switch (event->evt_id) {
    case BLE_PEER_MANAGER_ON_CONN_OPENED_PERIPHERAL: {
      #if APP_LOG_ENABLE
      . . .
      #endif // APP_LOG_ENABLE
      on_connection_opened_with_initiator
                    (event->connection_id);
    }
    break;
    case BLE_PEER_MANAGER_ON_CONN_CLOSED:
      . . .
      on_connection_closed(event->connection_id);
      break;
```

The following method creates a new connection to an initiator by calling `cs_reflector_create`. Note that the current implementation of the reflector can communicate with up to four initiators at the same time:

```
static void on_connection_opened_with_initiator
              (uint8_t conn_handle) {
  sl_status_t sc;
  . . .
  sc = cs_reflector_create
      (conn_handle, &cs_reflector_config);
```

As with STM32, most of the code is provided by the semiconductor manufacturer and does not usually need to be changed in practice.

Avoiding Pitfalls When Setting Up the Devboard

Almost all microcontroller architectures have some "peculiarity" that causes problems for those switching from other platforms. In the case of Silicon Labs, there are several issues — annoyance number one is the base firmware on the modules (which works in tandem with the actual application).

Upon closer inspection, you'll notice that Simplicity Studio (as shown in **Figure 9**) informs you about the firmware version.

Mismatches in versions sometimes lead to undefined behavior. This is problematic because such differences are only displayed if you select the — actually outdated — *Silicon Labs ARM Program* debugger configuration in *Debug Configuration* options.

Clicking the upgrade link also ends in an error message, but this is irrelevant — after the next restart, Simplicity Studio will report the firmware version as updated.

Problem number two concerns the bootloader. Silicon Labs examples are generally configured for coexistence with a bootloader software.

This is frustrating for newcomers, as compiling the bootloader is a separate project and requires extensive configuration in Simplicity Studio, which we will not cover here for space reasons.

As a workaround, it is advisable to first flash the board using the demo configuration discussed above, and then flash the actual program code via the debug profile.

In this case, the bootloader delivered by *Run* remains in the board's flash memory, which also makes the later delivered binary file with the same start address functional.

To execute successfully, then use *Start* from the *GDB SEGGER J-Link Debugging* debugger configuration again. Normally, there will be a breakpoint at the `sl_main_init()` method, which should be skipped when starting:

```
int main(void)
{
  // Initialize Silicon Labs device, system,
  // service(s) and protocol stack(s).
  // Note that if the kernel is present,
  // the start task will be started and software
  // component initialization will take place there.
  sl_main_init();
```

On a smartphone equipped with a relatively up-to-date Bluetooth LE stack, the other side can be found using a Bluetooth LE scanner as shown in **Figure 10** — even if the phone itself is not capable of channel sounding.

Careful examination of the sample code shows that Silicon Labs has placed log statements built according to the following scheme at various points:

```
void app_init(void)
{
  app_log_info(APP_LOG_NL);
  app_log_info("+-[CS Reflector by Silicon Labs]------
------------------+" APP_LOG_NL);
  app_log_info("+--------------------------------------
------------------+" APP_LOG_NL);
```

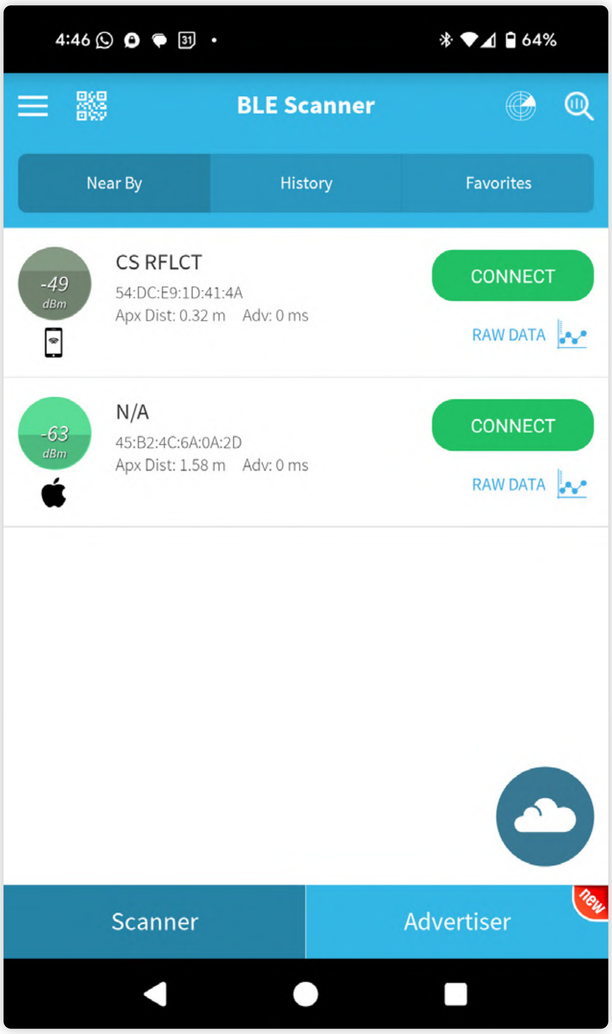


Figure 10: This reflector works perfectly.

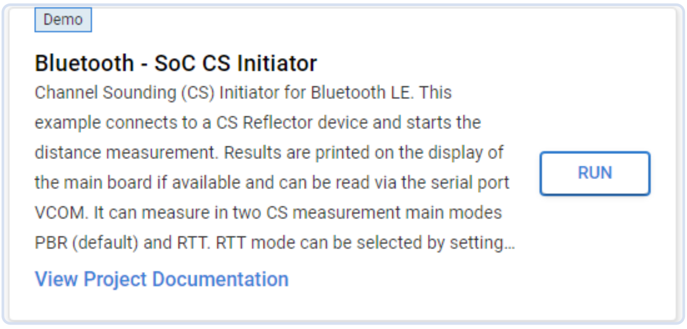
To make the messages visible, you first need a terminal program — the author likes to use TeraTerm on his Windows workstation. After starting the application, connect to the COM port assigned to the Silicon Labs test board in Device Manager.

In some cases, line breaks may not work correctly. In this case, click *Setup* → *Terminal*, then set the value under *Receive* in the *New-line* section to *AUTO*. As a result, the status messages become visible in TeraTerm.

Starting the Device Intended for Positioning

After starting the reflector, we need to set up the device responsible for determining the distance between the two devices. First, unplug board A and power it, for example, using a power bank (mind the standby current) or a power supply.

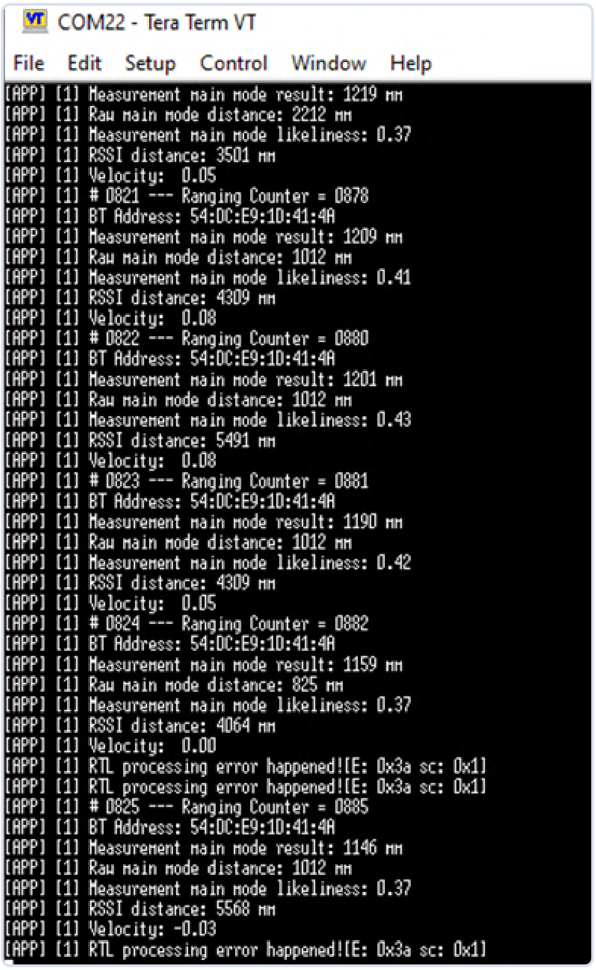
In the next step, it is advisable to reactivate the Launcher perspective, which provides the *Project Generation Experience* discussed above. A newly connected EFR32 developer kit should be recog-


Demo

Bluetooth - SoC CS Initiator

Channel Sounding (CS) Initiator for Bluetooth LE. This example connects to a CS Reflector device and starts the distance measurement. Results are printed on the display of the main board if available and can be read via the serial port VCOM. It can measure in two CS measurement main modes PBR (default) and RTT. RTT mode can be selected by setting...

Figure 11: If you run a demo program from the Demo tile, the bootloader is automatically installed.



```
[APP] [1] Measurement main mode result: 1219 mm
[APP] [1] Raw main mode distance: 2212 mm
[APP] [1] Measurement main mode likeliness: 0.37
[APP] [1] RSSI distance: 3501 mm
[APP] [1] Velocity:  0.05
[APP] [1] # 0821 --- Ranging Counter = 0878
[APP] [1] BT Address: 54:DC:E9:1D:41:4A
[APP] [1] Measurement main mode result: 1209 mm
[APP] [1] Raw main mode distance: 1012 mm
[APP] [1] Measurement main mode likeliness: 0.41
[APP] [1] RSSI distance: 4309 mm
[APP] [1] Velocity:  0.08
[APP] [1] # 0822 --- Ranging Counter = 0880
[APP] [1] BT Address: 54:DC:E9:1D:41:4A
[APP] [1] Measurement main mode result: 1201 mm
[APP] [1] Raw main mode distance: 1012 mm
[APP] [1] Measurement main mode likeliness: 0.43
[APP] [1] RSSI distance: 5491 mm
[APP] [1] Velocity:  0.08
[APP] [1] # 0823 --- Ranging Counter = 0881
[APP] [1] BT Address: 54:DC:E9:1D:41:4A
[APP] [1] Measurement main mode result: 1190 mm
[APP] [1] Raw main mode distance: 1012 mm
[APP] [1] Measurement main mode likeliness: 0.42
[APP] [1] RSSI distance: 4309 mm
[APP] [1] Velocity:  0.05
[APP] [1] # 0824 --- Ranging Counter = 0882
[APP] [1] BT Address: 54:DC:E9:1D:41:4A
[APP] [1] Measurement main mode result: 1159 mm
[APP] [1] Raw main mode distance: 825 mm
[APP] [1] Measurement main mode likeliness: 0.37
[APP] [1] RSSI distance: 4064 mm
[APP] [1] Velocity:  0.00
[APP] [1] RTL processing error happened![E: 0x3a sc: 0x1]
[APP] [1] RTL processing error happened![E: 0x3a sc: 0x1]
[APP] [1] # 0825 --- Ranging Counter = 0885
[APP] [1] BT Address: 54:DC:E9:1D:41:4A
[APP] [1] Measurement main mode result: 1146 mm
[APP] [1] Raw main mode distance: 1012 mm
[APP] [1] Measurement main mode likeliness: 0.37
[APP] [1] RSSI distance: 5568 mm
[APP] [1] Velocity: -0.03
[APP] [1] RTL processing error happened![E: 0x3a sc: 0x1]
```

Figure 12: The demo program reports the measured distances.

nized immediately. Then, again, go to *Example projects* and *Demos*, but this time select the project *Bluetooth - SoC CS Initiator*.

As with the reflector, the initiator's evaluation board must also be equipped with a bootloader. For this reason, in the first step, we select the *Demo* tile shown in **Figure 11** — clicking the *Run* button ensures both bootloader and demo program are transferred to the target board.

In the next step, open TeraTerm again, where distance information is already visible, as shown in **Figure 12**.

For a "detailed analysis" of the source code, the next step is to create a new project in Simplicity Studio based on the template. The necessary procedure is exactly as discussed above — in the following steps, we'll use the name *bt_cs_soc_initiator* and adopt the remaining project assistant settings as previously outlined.

In the *app.c* file, the following two variables are required or interesting, as they are used to store parameters and memory fields necessary for the initiator function:

```
static cs_initiator_config_t
  initiator_config = INITIATOR_CONFIG_DEFAULT;
static cs_initiator_instances_t
  cs_initiator_instances[CS_INITIATOR_MAX_CONNECTIONS];
```

The actual configuration of the various parameters takes place in the method `void app_init(void)`. An application timer is also set up there, which activates the following callback once per second:

```
static void app_timer_callback
     (app_timer_t *timer, void *data) {
  (void)timer;
  (void)data;
  cs_initiator_display_update();
}
```

The processing of measured data takes place in the method `void app_process_action(void)`, which interprets the information according to the following scheme:

```
void app_process_action(void)
{
  for (uint8_t i = 0u; i < CS_INITIATOR_MAX_CONNECTIONS;
               i++) {
    if (cs_initiator_instances[i].measurement_arrived) {
      // write results to the display & to the iostream
      cs_initiator_instances[i].measurement_arrived =
                                          false;
      log_info(APP_INSTANCE_PREFIX "# %04lu
              --- Ranging Counter = %04lu" NL,
           cs_initiator_instances[i].conn_handle,
           cs_initiator_instances[i].measurement_cnt,
           cs_initiator_instances[i].ranging_counter);
```

Interestingly, the `Ranging Counter` is a counter that quantifies the number of "completed" measurement processes.

It is also interesting to ask how the interaction between the Bluetooth stack and user code works. As with the reflector, the main part of the code is event-driven. The entry point for events occurring in the Bluetooth stack is the following method:

```
void sl_bt_on_event(sl_bt_msg_t * evt)
{
  sl_status_t sc;
  uint8_t instance_num;
  const char* device_name =
    REFLECTOR_DEVICE_NAME;

  switch (SL_BT_MSG_ID(evt->header)) {
```

The initiator instance is then activated using the various predefined methods. Notably, as shown below, this involves a relatively large amount of housekeeping code, which, among other things, manages the update of several initiator instances:

```
static sl_status_t create_new_initiator_instance
                         (uint8_t conn_handle)
{
  . . .
  // Store the new initiator instance
  for (uint32_t i = 0u; i <
       CS_INITIATOR_MAX_CONNECTIONS; i++) {
    if (cs_initiator_instances[i].conn_handle ==
           SL_BT_INVALID_CONNECTION_HANDLE) {
      cs_initiator_instances[i].conn_handle = conn_handle;
       . . .
  }
  sc = cs_initiator_create(conn_handle,
                           &initiator_config,
                           &rtl_config,
                           cs_on_result,
                           cs_on_intermediate_result,
                           cs_on_error,
                           NULL);
```

Analysis with Flowchart

The invocations and messages exchanged between the different components of the Bluetooth stack are relatively complex. Silicon Labs offers a compilation at URL [4], which includes, among other things, clear flowcharts.

Conclusion

Using the methods specified by the Bluetooth SIG for indoor positioning is recommended, as the systems can then work as part of an open ecosystem with devices from third-party manufacturers. With the EFR32xG24 platform from Silicon Labs, a powerful and cost-effective work environment is available and can be purchased immediately from various distributors.

250335-01

About the Author

Ing. Tam Hanna has been involved with electronics, computers, and software for more than 20 years. He is a freelance developer, book author, and journalist (www.instagram.com/tam.hanna). In his free time, Tam enjoys working with 3D printing and also distributes cigars.

Questions or Comments?

If you have technical questions or comments about this article, please contact the author (tamhan@tamoggemon.com) or the Elektor editorial team at editor@elektor.com.

WEB LINKS

[1] Simplicity Studio, Silicon Labs: https://www.silabs.com/developer-tools/simplicity-studio
[2] Simplicity Studio SDK, GitHub: https://github.com/SiliconLabs/simplicity_sdk
[3] Direction Finding Methods, Silicon Labs: https://tinyurl.com/silabs-direction-finding
[4] Sample Applications, Silicon Labs: https://tinyurl.com/silabs-sample-applications

Powering the Future of Wireless Communication

BTRY's Ultra-Thin Solid-State Batteries

Contributed by BTRY

As wireless communication becomes central to modern technology, energy solutions must evolve to match. BTRY has developed an innovative solid-state battery designed to meet the specific demands of wireless, mobile, and connected devices.

At just 0.1 mm thick, BTRY's batteries are the thinnest batteries in the world. They are non-flammable, mechanically robust, and operate safely in extreme temperatures from –40°C to +150°C. BTRY's technology combines the advantages of batteries and super-capacitors with high energy and power density. Its ultra-low self-discharge rate of <1% per year enables maintenance-free operations and long device lifetimes, while its rapid one-minute charging and discharging supports power peaks up to 500 mA, ideal for transmitting wireless signals.

BTRY's batteries can be cut to any shape desired or even rolled into a cylinder, enabling simple integration into a wide range of devices. Applications range from wireless access cards and smart wearables to maintenance-free IoT sensors and asset trackers.

With wireless devices becoming thinner, more autonomous, and more connected, BTRY provides the missing power link: compact, fast, and safe energy storage for edge communication.

Contact us to learn more: info@btry.ch | www.btry.ch

50600-01

BTRY

www.btry.ch
info@btry.ch

The thinnest battery in the world

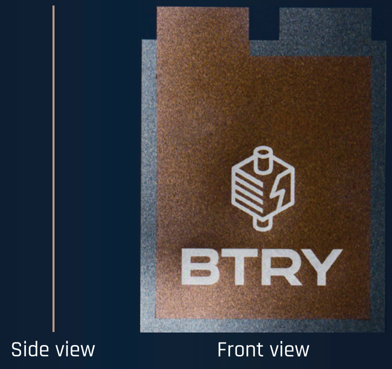
BTRY powers the next generation of wireless devices with ultra-thin solid-state batteries.

Small battery, endless possibilities

0.1 mm thick
Integrated energy storage solutions in ultra-thin devices.

Charges and discharges in 1 minute
High power peaks for wireless communication, without a capacitor.

Stable from -40 °C to 150 °C
Ensuring reliable performance across extreme temperatures.



Test-Driven Development in **Firmware Writing**

By Yahya Tawil (Turkey)

Embedded systems engineers love seeing their code run successfully on hardware on the first attempt. However, in reality, bugs and mistakes often make this challenging. What if there was a way to catch issues before they happen? In this article about test-driven development, we will explore a technique to help write code with fewer bugs and unintended errors.

Since firmware development is a branch of software engineering, it has adopted the Test-Driven Development (TDD) approach from software engineering, which emphasizes writing tests before the code itself. However, some firmware developers follow a Test-Later Development (TLD) approach, where testing — especially unit tests — is postponed until the code is considered functional. While this mindset may seem efficient and natural, it often leads to poor test coverage. As a result, the codebase becomes more vulnerable to integration failures when new features are introduced.

The great thing about TDD is that it ensures full test coverage when followed strictly. Developers adopt a test-first mindset, writing tests before the actual code. Initially, the test fails since no implementation exists (*Red*), see **Figure 1**. Next, the code is written to implement the unit and pass the test (*Green*). Finally, the code is refined and optimized while keeping the test successful (*Refactor*). This cycle — Red, Green, Refactor — is the core of TDD.

TDD flips the traditional firmware development process, making it seem unfamiliar and challenging at first. A common concern is how to apply unit testing to firmware running on a microcontroller, given its tight dependency on vendor SDKs and toolchains. This article explores these challenges and clarifies how TDD can be effectively integrated into embedded development.

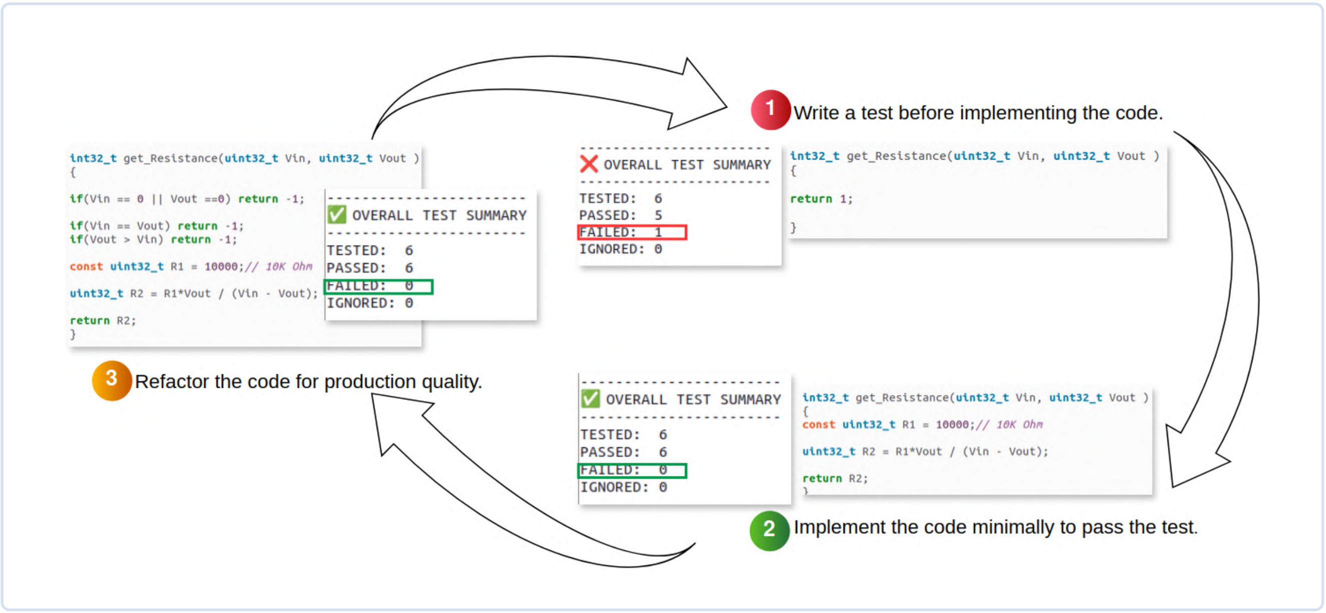
Whether you've heard previously about TDD from a book, colleague, or a talk — or, if you are completely new to the concept — this article is for you.

Understanding Unit Tests in Embedded Systems

TDD cannot be introduced without a clear understanding of **unit tests**. While this article is not meant to teach how to write unit tests, it's important to clarify some key terms.

The term **test double** is widely used in unit testing. It refers to objects that replace real system components to isolate the System Under Test (SUT) from dependencies. In embedded systems, using test doubles is essential, as dependencies on SDKs and third-party libraries can be more restrictive than in general software development. Test doubles are especially critical in off-target testing, where the host compiler (usually on a PC) is used to generate binaries, and neither the SDK nor real MCU peripherals are involved.

Test doubles come in different types, with the most common being Fakes, Stubs, and Mocks:


1 Write a test before implementing the code.
2 Implement the code minimally to pass the test.
3 Refactor the code for production quality.

```
----------------------
❌ OVERALL TEST SUMMARY
----------------------
TESTED:  6
PASSED:  5
FAILED:  1
IGNORED: 0
```

```
int32_t get_Resistance(uint32_t Vin, uint32_t Vout )
{

return 1;

}
```

```
----------------------
✅ OVERALL TEST SUMMARY
----------------------
TESTED:  6
PASSED:  6
FAILED:  0
IGNORED: 0
```

```
int32_t get_Resistance(uint32_t Vin, uint32_t Vout )
{
const uint32_t R1 = 10000;// 10K Ohm

uint32_t R2 = R1*Vout / (Vin - Vout);

return R2;
}
```

```
int32_t get_Resistance(uint32_t Vin, uint32_t Vout )
{

if(Vin == 0 || Vout ==0) return -1;

if(Vin == Vout) return -1;
if(Vout > Vin) return -1;

const uint32_t R1 = 10000;// 10K Ohm

uint32_t R2 = R1*Vout / (Vin - Vout);

return R2;
}
```

```
----------------------
✅ OVERALL TEST SUMMARY
----------------------
TESTED:  6
PASSED:  6
FAILED:  0
IGNORED: 0
```

Figure 1: An illustration of TDD iteration steps.

- **Fakes** provide a simplified implementation of a dependency, making testing easier. For example, replacing a NOR Flash key/value store with an in-memory key/value store on the host during off-target testing.
- **Stubs** return predefined, hardcoded responses with minimal logic. For example, replacing an ADC reading function with one that always returns a fixed value.
- **Mocks** verify interactions, ensuring functions are called with expected arguments. Unlike stubs, mocks act as spies on function calls. For example, when testing an I^{2}C sensor driver, a mock can check if the correct register address is sent, whereas a stub would remain passive.

It's important to note that the differences between test doubles run deeper, but if you're unfamiliar with these concepts, this explanation should be a good starting point. Additionally, dependency replacement in C/C++ can be achieved using different test double techniques. These include: Interface-Based Replacement, Inheritance-Based Replacement, Composition-Based Replacement, Link-Time Replacement, Macro Preprocessed Replacement, and Vtable-Based Replacement. More on this in [1].

Unit tests require a framework consisting of a library and a test runner. The library provides assertions and support for test doubles like mocks, while the test runner executes the tests. It is worth mentioning that not all frameworks have built-in support for Mocks. The framework also manages test fixtures, typically called setup and teardown, which run before and after each test to ensure a controlled environment.

TDD in Practice

TDD follows an iterative loop:

1. Write a test before implementing the code. This ensures the function's definition exists but lacks a body. The test suite, including previously passing tests, should run, with the new test failing initially.
2. Implement the code minimally to pass the test. This first version may not be optimal, but running the full test suite prevents regressions.
3. Refactor the code for production quality. Running tests again ensures no new faults emerge. This approach saves time compared to the later debugging approach.

We'll use a demo project to demonstrate the TDD loop in practice. While you can use any unit testing framework, such as Unity or GoogleTest, we'll go with Ceedling this time. Ceedling is user-friendly and comes with built-in mocking capabilities via CMock.

In this demo project, we'll build a controllable voltage divider using a digital potentiometer, such as the AD5160 [2] (a 256-position, 10 kΩ end-to-end resistance SPI digital potentiometer), connected via SPI (**Figure 2**). As a first step, we can break the system into two components: one for calculating the required resistance to achieve the desired output voltage and another for sending this value to the chip over SPI. To keep the focus on demonstrating TDD, we'll make several simplifications.

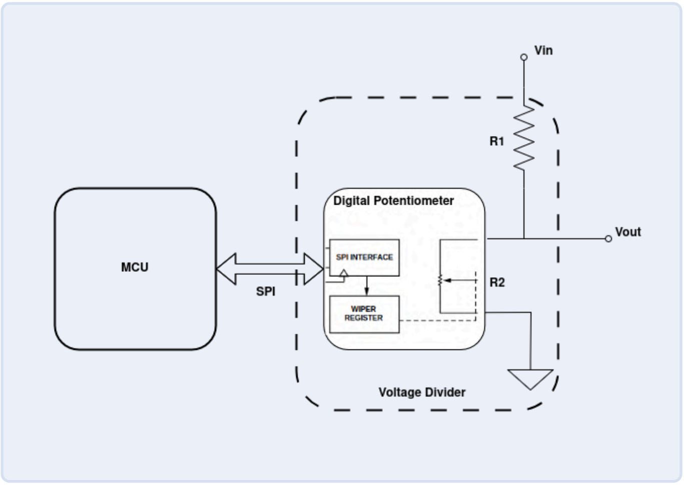


Figure 2: Software-Controlled Voltage Divider.

After installing Ceedling [3], we need to create the project using the command:

```
ceedling new sw_voltage_divider
```

Then we create a module using the command:

```
ceedling module:create[voltageDiv]
```

This module will be the application code that handles calculating the required resistance value. This will create the following directory structure.

```
├── project.yml
├── src
│   ├── voltageDiv.c
│   └── voltageDiv.h
└── test
     └── test_voltageDiv.c
```

Calculating Resistance

Our first step is to implement the code for calculating the required resistance. This will be a simple function that takes two parameters — input voltage and desired output voltage — and returns the calculated resistance value. We'll begin by writing the first unit test to verify this function's output. Since this test will initially fail, it marks the starting point of our TDD cycle.

```
// in src/voltageDiv.h
int32_t get_Resistance(uint32_t Vin, uint32_t Vout );

// in src/voltageDiv.c
int32_t get_Resistance(uint32_t Vin, uint32_t Vout )
{
  return -1;
}

// in test/test_voltageDiv.c
void test_whenValidVoutAndVinProvided_
        thenReturnCorrectResistance(void)
{
  TEST_ASSERT_EQUAL_INT32(get_
                      Resistance(5000,2500),10000);
     // Assuming R1 must be 10 k, when Vin = 5 V and
     // Vout = 2.5 V
}
```

The `TEST_ASSERT...` function will check that we get back `10000` in `int32` format, if we call the `get_Resistance` function with the given parameters.

The initial implementation will calculate the resistance using the voltage division rule. This will be sufficient to pass the first test.

```
// in src/voltageDiv.c
int32_t get_Resistance(uint32_t Vin, uint32_t Vout )
{
  return 10000*Vout / (Vin - Vout);
}
```

Next, we'll refactor the code by defining R1 as a macro in the header file and adding a check to ensure Vin and Vout are not equal, preventing division by 0. Additionally, we'll introduce guard conditions to ensure neither Vin nor Vout is 0 and that Vin is always greater than Vout. Running the unit tests will help confirm that our refactoring hasn't introduced any errors.

```
// in src/voltageDiv.c
int32_t get_Resistance(uint32_t Vin, uint32_t Vout )
{
  if(Vin == 0 || Vout == 0) return -1;

  if(Vin == Vout) return -1;
  if(Vout > Vin) return -1;

  uint32_t R2 = R1*Vout / (Vin - Vout);

  return R2;
}
```

The potentiometer driver

Now, let's build the potentiometer driver. We'll create a function that sets the potentiometer to the desired resistance value by sending it over SPI to the chip. The corresponding unit test will verify this behavior, and as expected in TDD, the test will initially fail on the first run. Let's create a module for it using `ceedling module:create[AD5160]`.

```
// in src/AD5160.h
uint8_t pot_set(uint32_t res_value);

// in src/AD5160.c
uint8_t pot_set(uint32_t res_value)
{
  return 0;
}
// in test/test_AD5160.c
void test_AD5160_SetResistanceViaSpi(void)
{
  TEST_ASSERT_EQUAL_INT8(pot_set(5000),1);
}
```

During normal operation, the pot_set function will return 1 to confirm that the potentiometer was successfully set; this is checked by the TEST_ASSERT... function. As you can see, the initial test will fail.

The basic implementation of the pot_set function involves selecting the appropriate number of steps for the digital potentiometer to achieve a resistance value close to the desired one. Since this would require an SPI transfer with actual hardware, we'll use a mock to replace this actual SPI transfer while running the tests. In Ceedling, this is done by calling spi_transfer_ExpectAndReturn before the assertion that tests pot_set. All we need to do is include the spi_transfer declaration in *SPI.h*.

```
// in test/test_AD5160.c
void test_AD5160_SetResistanceValue(void)
{
  spi_transfer_ExpectAndReturn(128,1);
     // SPI Mock used inside pot_set
  TEST_ASSERT_EQUAL_INT8(pot_set(5000),1);
}
// in src/AD5160.c
uint8_t pot_set(uint32_t res_value)
{
  uint8_t step = 10000/256;
  uint8_t D = res_value / step;
  return spi_transfer(D);
}
```

Finally, we can refactor the code by adding macro definitions and implementing a check to ensure the required resistance does not exceed the potentiometer's maximum value.

```
uint8_t pot_set(uint32_t res_value)
{
  if(res_value > R2_MAX) return 0;

  uint8_t D = res_value / POT_RESOLUTION ;

  return spi_transfer(D);
}
```

To view the full source code for the demo project, visit the GitHub repository [4].

Evaluating the Efficiency of TDD

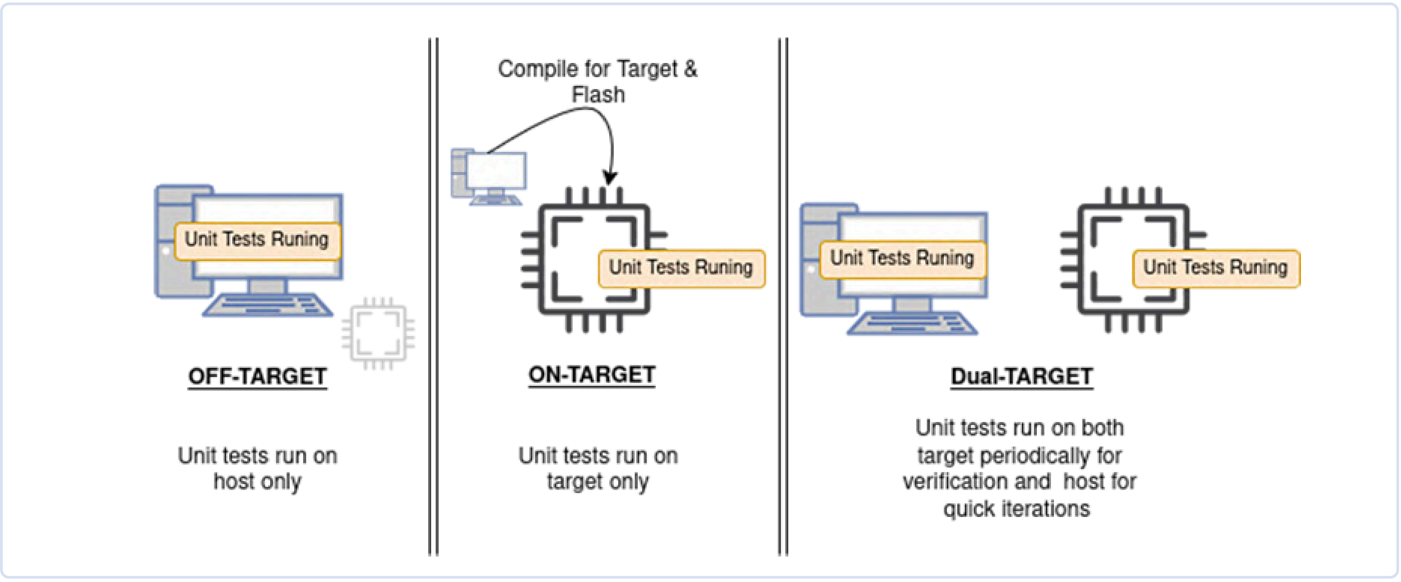
Like any engineering approach, TDD isn't always the optimal solution for every scenario. However, even if not fully applied, its principles can greatly influence the way we think about software development. TDD offers significant benefits, such as:

- TDD encourages modularity and dependency isolation, which are essential for effective unit testing. As a result, your code becomes well-structured and portable. For example, it discourages mixing application logic with direct hardware interactions, even in the name of efficiency.
- The TDD iteration keeps developers focused on one issue at a time while emphasizing external behavior and interface design to create effective unit tests.
- Running unit tests at each step helps quickly localize errors and provides immediate feedback.
- TDD enables development to begin even before the target hardware is available by using mocks to simulate hardware components, allowing testing and validation early in the process.
- TDD allows development to proceed independently, even if colleagues haven't finished their parts. For example, if a driver is still under development, you can use mocks to simulate its behavior.
- The TDD iteration benefits developers by ensuring steady development. Having one or more unit tests covering a feature confirms its successful implementation.

One of the few scientific studies on applying TDD in embedded software development, "Test-Driven Development and Embedded Systems: An Exploratory Investigation" [5] examined its impact on software quality and developer productivity. In the study, nine Master's students implemented trial projects using both TDD and non-TDD approaches. The results showed an increase in external quality but no significant difference in productivity. While this study doesn't provide a definitive answer, its findings are worth noting.

There's no need to highlight the obvious challenges of TDD, such as the extra code required for unit tests and the effort needed to replace dependencies with test doubles like mocks, which can sometimes be complex. The decision to use TDD should always weigh the added effort against its potential benefits.

Finally, it's worth mentioning that TDD is a development methodology, not an implementation strategy — it won't teach developers how to structure or architect their code. TDD aligns with Agile, particularly Extreme Programming (XP), which advocates frequent releases in short development cycles and includes a test-first approach. However, TDD itself is not exclusive to XP. Additionally, it's important to note that unit tests written during TDD should be complemented by system and integration testing.



Unit Tests Runing
OFF-TARGET
Unit tests run on host only
Compile for Target & Flash
Unit Tests Runing
ON-TARGET
Unit tests run on target only
Unit Tests Runing
Unit Tests Runing
Dual-TARGET
Unit tests run on both target periodically for verification and host for quick iterations

Figure 3: TDD On-Target, Off-Target, and Dual-target strategies.

On-Target vs. Off-Target Testing: Which One to Choose?

Off-target and on-target testing refer to where tests are executed (see **Figure 3**). Off-target testing runs on the host machine used for firmware development, while on-target testing runs directly on the hardware that will execute the final firmware. This involves running unit tests on the target hardware while relaying test results to the host, typically through UART logs or a similar communication protocol.

Some may argue that off-target TDD is inefficient for testing real hardware functionality, and while this is partially true, it highlights the need for on-target or dual-target testing in some cases. However, off-target TDD can be crucial in scenarios where hardware failures are difficult to trigger or reproduce. For example, when developing a flash memory driver, testing failure cases like SPI bus errors or flash memory faults is challenging because the hardware typically operates correctly under normal conditions.

On-target testing may seem more preferred and realistic, but there are cases where off-target testing is more practical:

- Target is still under development and not yet ready for use.
- Limited memory space on the target prevents running all unit tests.
- No debugging ports or a proper output interface to display test results.
- Limited hardware availability, making it difficult for all team members to test.
- Certain errors (e.g., bus errors) cannot be reliably triggered on real hardware.
- The test suite is taking too long due to the time required for flashing binaries and retrieving test results.

In such cases, off-target TDD helps maintain development momentum while periodic or partial on-target testing could be done when needed, referred to as dual-target.

Some developers think unit test frameworks like Unity work only on the host, but that's not true. Unity can be adapted to run directly on the target, with test results printed via interfaces like UART. However, on-target testing is harder to automate.

On the other hand, off-target testing carries risks, such as differences between the host and target compiler toolchains. For example, an int is typically 4 bytes on most ARM Cortex-M targets but can be 4 or 8 bytes on x86-64. A great way to get the best of both worlds is to apply TDD while running the code with the target's toolchain in a fast and seamless manner using emulators like QEMU.

TDD Through the Eyes of the Firmware Developer Community

Gathering insights from the developer community provides valuable feedback. Online platforms like Reddit offer a way to hear from industry professionals, including those from companies that typically don't disclose their use of TDD. Since anonymous employees often share real-world experiences, I've selected a few opinions worth highlighting, without adding personal judgment.

One developer believes TDD is totally worth it, acknowledging that it may feel uncomfortable at first and isn't the fastest way to develop software. Another emphasizes that with proper mocking and faking, unit testing can be done with zero hardware interaction.

Another developer expressed resistance to TDD, especially in projects with many unknowns that only become clear during development. In their view, this leads to wasted time deleting and adapting tests. They described a common approach: prototyping the code until it reaches a stable state with successful high-level functional or integration tests. Only then do they aggressively refactor and add unit tests to key areas that won't change frequently.

Another useful TDD use case mentioned by a developer is in multi-stage signal analysis algorithms, making each stage testable separately. Additionally, the unit tests resulting from following TDD can help detect rare issues on the host, such as hardware failures and timing problems, which might be difficult to reproduce on the actual hardware.

The Bottom Line

Some advocate TDD without addressing its pros and cons, while others reject it entirely. However, learning and experimenting with TDD can enhance development practices even if not followed strictly. TDD is particularly useful for large projects with multiple team members and complex functionality, but it also benefits smaller teams and projects.

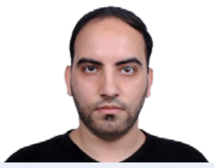
Developers who resist TDD may struggle with firmware architecture design or writing effective test cases. Actually, a lot of the misunderstandings about TDD may come from a lack of experience in strong writing unit tests. Approaches like "test interface, not implementation" and Behavior-Driven Development (BDD) can help minimize redundant test rewriting. Additionally, mutation testing evaluates test quality by introducing small code changes and checking if tests detect them.

Finally, the usual further reading recommendation is the foundational James Grenning's book *Test-Driven Development for Embedded C* [6], which, at the time of writing this article, is the only book specifically focused on applying TDD in embedded systems. Additionally, I recommend reading the book chapter "Test-Driven Development as a Reliable Embedded Software Engineering Practice" from *Embedded and Real-Time System Development: A Software Engineering Perspective* [7], a book that provides an in-depth explanation of TDD in embedded systems and suggests valuable enhancements to it. A vast number of resources were used to prepare this article and are listed in a GitHub *README* file [8] for space-saving purposes.

250092-01

Questions or Comments?

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



About the Author

Yahya Tawil is a seasoned embedded systems engineer with a decade of practical experience and a Master's degree in Electronics and Computer Engineering. He works as a consultant at Atadiat-Lab and as a freelancer. Embedded systems are both his hobby and profession. His experience mainly includes firmware and electronics design.

WEB LINKS

[1] J. Boydens, "Test-driven development as a reliable embedded software engineering practice," Academia.edu: https://www.academia.edu/download/45157291/302011_1_En_4_Chapter_OnlinePDF.pdf
[2] AD5160 Digital Potentiometer Data Sheet, Analog Devices: https://www.analog.com/media/en/technical-documentation/data-sheets/ad5160.pdf
[3] Ceedling Build System for Growing Robust C Projects, ThrowTheSwitch.org: https://www.throwtheswitch.org/ceedling
[4] TDD demo on GitHub: https://github.com/yahyatawil/TDD-demo-sw-voltage-divider
[5] M. Esposito, S. Romano and G. Scanniello, "Test-Driven Development and Embedded Systems: An Exploratory Investigation," 2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA): https://ieeexplore.ieee.org/document/10371442/
[6] James W. Grenning, "Test-Driven Development for Embedded C," The Pragmatic Programmers, 2011 : https://pragprog.com/titles/jgade/test-driven-development-for-embedded-c
[7] M. A. Khan et al., "Embedded and Real Time System Development: A Software Engineering Perspective," Springer, 2016: https://link.springer.com/book/10.1007/978-3-642-40888-5
[8] ES-TDD-Resources on GitHub: https://github.com/yahyatawil/ES-TDD-Resources

Phone-Controlled Model Car

Wi-Fi + ESP32 + Smartphone = Remote Control

By Huub Zegveld (The Netherlands)

Can a smartphone really control a car? With an ESP32 and some clever engineering, a model car becomes a test bed for exploring this idea — from wiring to coding to real-world results.

Is it possible to control a car using a smartphone? That question intrigued me. Since most people carry a smartphone everywhere, it would be convenient to use one as a remote control. As a proof of concept, I added an ESP32 to a model car, wired the components, wrote the software, and conducted some tests. Here's what I discovered.

EP32-Based Solution

As mentioned, I chose an ESP32 module for its built-in microcontroller and Wi-Fi capabilities. Connecting the ESP32 to a local Wi-Fi network is straightforward. Setting up a simple webserver to accept remote control commands via a web interface is also relatively easy — more on this below.

For this project, I designed a PCB in KiCad v6 (see **Figure 1**). The ESP32 module can be plugged onto this board. The module I used features a USB-C connector, a printed Wi-Fi antenna, and 2 rows of 15-pin headers (see **Figure 2**). One note for those who want to build this project and use my PCB: Other ESP32 modules may also fit — but the pin compatibility and the distance between the headers should be checked in advance.

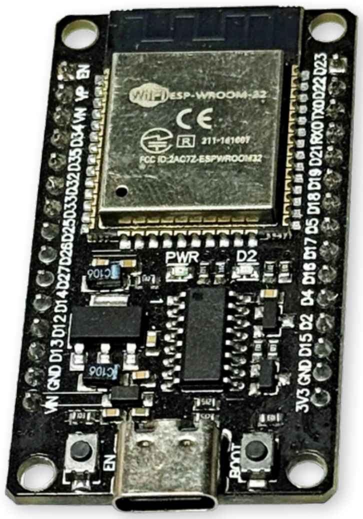


Figure 2: The right ESP32 BoB with 30 pins. You can use variants with USB-C as shown or cheaper ones with a micro USB connector.

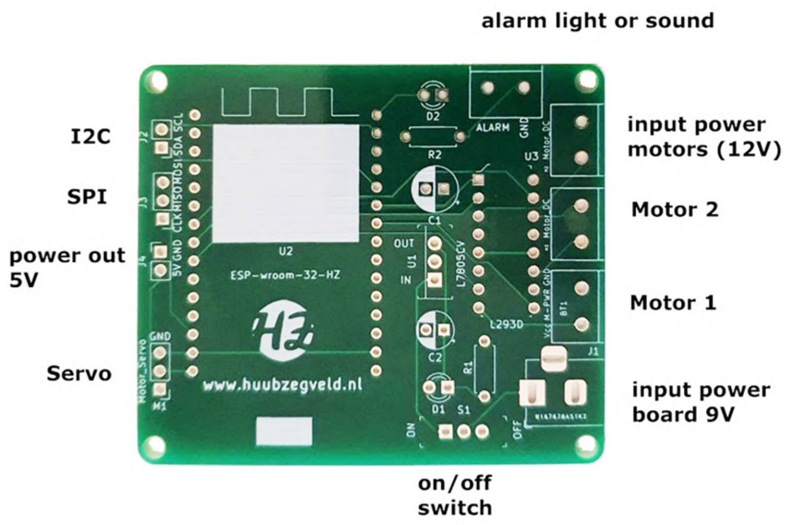


Figure 1: The PCB designed for the model car.

I designed not only the electronics and software, but also the mechanical frame of the car myself. I used FreeCAD 0.21 [1] (note: version 1.0 is now available) and printed the chassis using UltiMaker Cura [2]. **Figure 3** shows the CAD model of the frame.

During testing, I found it inconvenient to check my router's settings or connect the ESP32 to a PC via USB just to discover its IP address. A more elegant solution was to add a small OLED display to the car to show

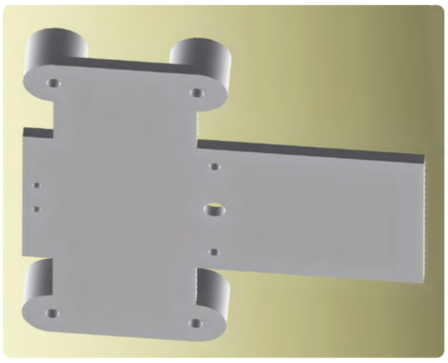


Figure 3: The CAD-designed frame of the model car.

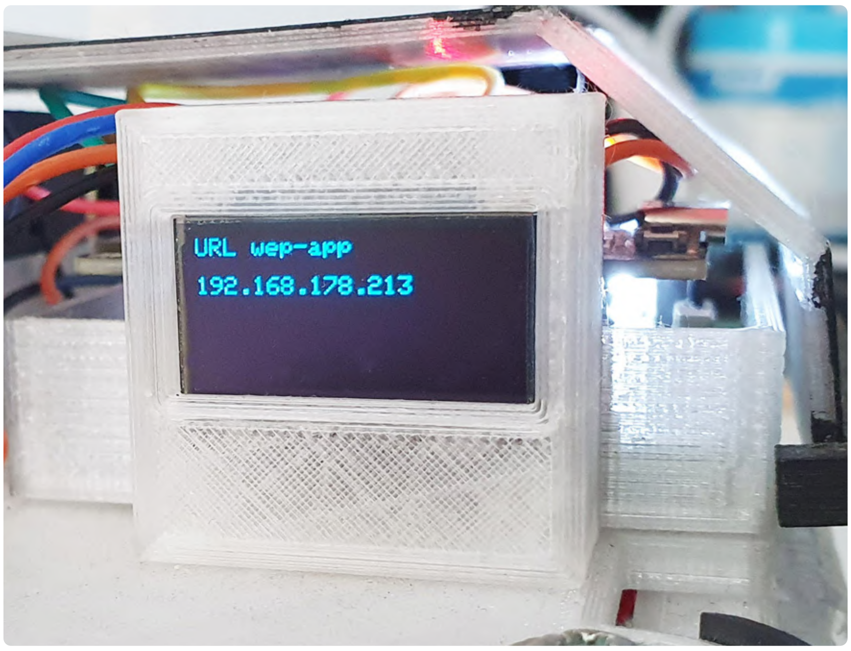


Figure 4: The 0.96" OLED display in a 3D printed housing shows the IP address the ESP32 obtains from the router's DHCP server.

the assigned IP. I also wanted features like collision avoidance and lighting, so I included an ultrasonic sensor and LEDs.

Software

For the ESP32 code, I used Arduino IDE 2.3.6 [3] as the primary development environment. My initial attempts were with a 433-MHz RF module, and I was able to reuse code, such as *dc-motor.h*. I did face some issues with ESP32 PWM functions, but after online research, I could resolve them.

The code for *dc-motor.h* and *dc-motor.cpp* was originally written in CodeBlocks IDE [4], and can be downloaded from [5]. I employed several key libraries:

- *WiFi.h* – for Wi-Fi connection and server setup
- *dc-motor.h* – for PWM setup and motor control
- *Adafruit_SSD1306.h*, *Wire.h*, *Adafruit_GFX.h* – for OLED integration

Smartphone Remote Control

No additional software is required on the smartphone — just a browser connected to the same Wi-Fi network as the car, respectively the ESP32. The ESP32 obtains an IP address from your router's DHCP server (e.g., *192.168.178.213*), which is shown on the OLED display (see **Figure 4**).

Entering this IP in the browser of the smartphone sends an HTTP request to the ESP32's webserver, which responds with a basic HTML control page (see **Figure 5**). Tapping a command like "stop" triggers another request, e.g., *http://192.168.178.213/CMD=stop*, which the ESP32 interprets to stop the car. That's all.

Hardware of the Car

For this project a usual BOM like the **Component List** may not be fully sufficient, so here are some key details: Besides the ESP32 module with 30 pins, a motor driver is required. For this, I chose the integrated L293D solution. Next is the OLED display: a low-cost module connected via I²C is enough. I opted for one with a standard 0.96" diagonal and a resolution of 128×64 pixels (type of the display controller should be SSD1306).

For collision detection, I included an HC-SR04 ultrasonic module, which is widely available at low cost. Since the electronics is powered by a 9 V source, a 7805 voltage regulator is needed to supply 5 V to the ESP32.

In addition to the electronic components, some mechanical parts are naturally required. As mentioned, the chassis is 3D-printed. For the wheels and motors, I repurposed parts

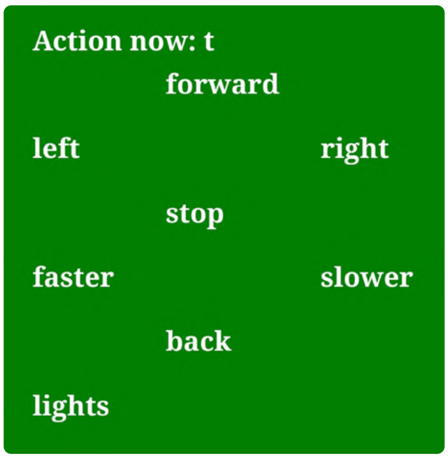

Action now: t
forward
left right
stop
faster slower
back
lights

Figure 5: This very simple web page appears on the smartphone screen to control the car via Wi-Fi.

Component List

Resistors
R1, R2 = 330 Ω*

Capacitors
C1 = 100 µ / 25 V, pitch 2.5 mm, ø 6.3 mm
C2 = 47 µ / 16 V, pitch 2.5 mm, ø 5 mm

Semiconductors
Display, OLED, 0.96", SSD1306*
Ultrasonic sensor HC-SR04
D1, D2 = LED*
U1 = 7805, TO220
U2 = ESP32 BoB, 30 pin*
U3 = L293D, DIL16

Miscellaneous
S1 = sliding switch, 1×3 pole, pitch 2.5 mm
J1 = barrel socket
J2, J4 = 2 pin vertical header, pitch 1/10"
J3, M1 = 3 pin vertical header, pitch 1/10"
2x 1×15 pin female vertical headers, pitch 1/10"
4x 2 pin screw connectors, pitch 5 mm
DIP16 socket for U3
Rotatable rear wheel*
2 driven front wheels
2 motors 12 V*
Chassis and mechanical parts*
Clip for 9 V battery
Case for 8 AA Batteries
PCB

* see text

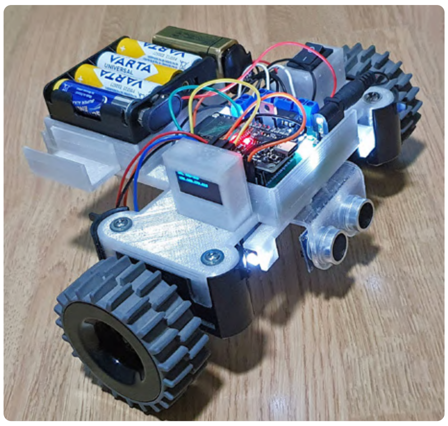


Figure 6: The open prototype shows nearly all of its components.

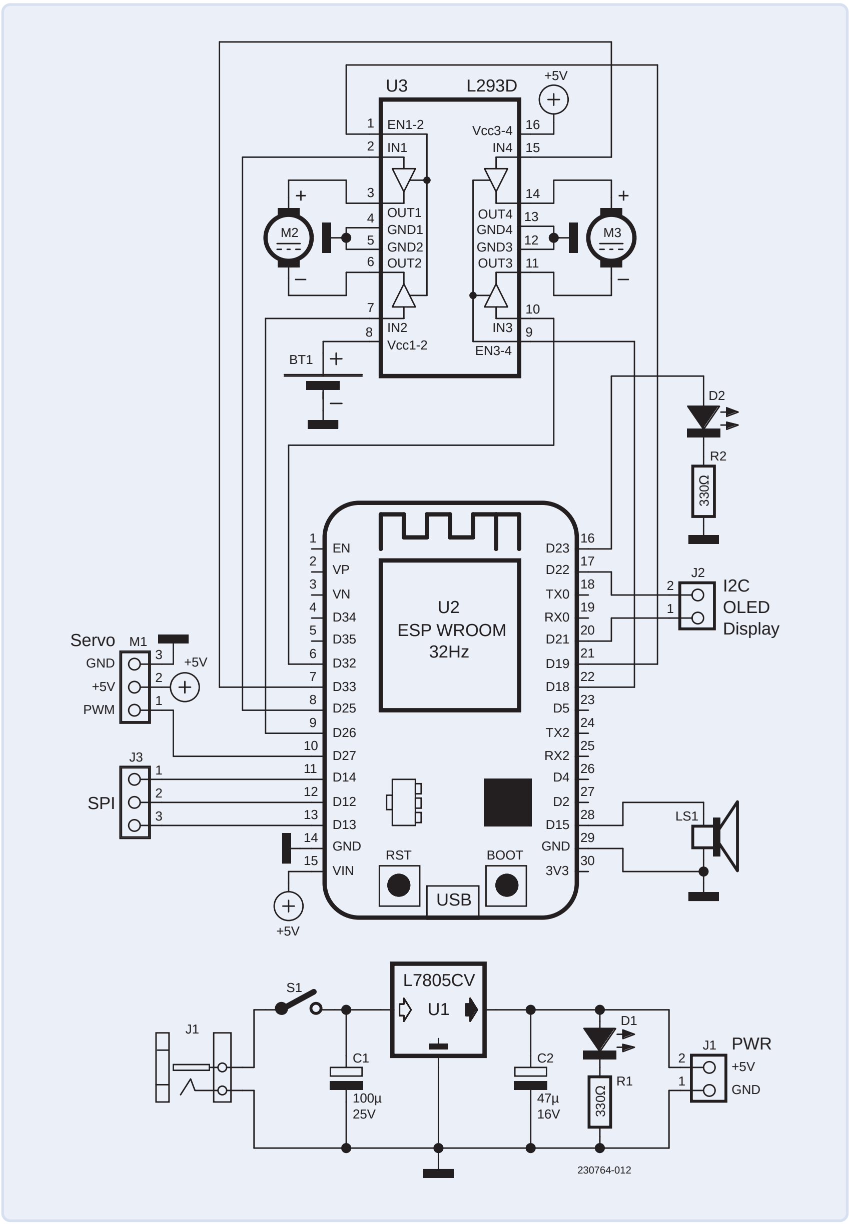


Figure 7: The car's electronic circuit is composed of an ESP32 Bob, a voltage regulator, and a motor driver IC.

from an old robot vacuum cleaner. The motors were of 12 V type, making the power supply straightforward. Naturally, bolts and nuts were also used. The completed model car is shown in **Figure 6**.

Circuit

The circuit diagram in **Figure 7** shows the interconnections of the ESP32 module (U2), the motor driver IC (U3), and the voltage regulator IC (U1), plus some passive components. As already mentioned, the motor driver IC U3 is powered by eight AA cells in series and therefore with 12 V. The voltage regulator IC U1 supplies 5 V — derived from 9 V – to the rest of the electronics. Notably, U1 and U3 require no heat sink, as power dissipation is low.

When designing the board, I initially planned to use a servo motor for steering. However, since it wasn't needed, I repurposed the power connection at connector M1 for the OLED display and left GPIO27 (Pin 10 of the ESP32 BoB) unused. J2 integrates the I²C connection for the display.

The LED D2 is used as switchable front light. You can connect a white LED here. If you prefer two LEDs, the second one needs an own series resistor like R2, and 2nd D2 plus 2nd R2 should be connected in parallel to D2/R2. LS1 can make a sound if the car detects an obstacle. However, it can be replaced by a red LED (plus series resistor) for a more visual feedback.

J3 integrates the digital input and output pins for the ultrasonic sensor (but could be used for an SPI connection also). The HC-SR04 ultrasonic sensor uses pulses that are inaudible to humans. The trigger pin (GPIO14, Pin 11 on ESP32) initiates a pulse, while the echo pin (GPIO12, Pin 12) signals receiving the reflected echo. The time delay is measured using the function `pulseIn(Echo,HIGH)`, and the distance is calculated via the formula:

distanceCm = duration * SOUND_SPEED / 2

If an object is detected within 20 cm, the car stops and reverses. For further details and example code, refer to the Handson Technology guide [6].

Assembly and Further Improvements

Software sources and the layout files for the PCB can be downloaded from the Elektor Labs website [5]. The populated board should look like my prototype in **Figure 8**. Because there

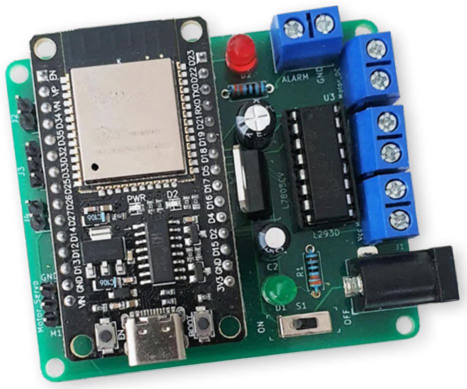


Figure 8: The fully populated board of the model car. Due to the low currents of the 7805 and the high efficiency of the L293D, neither ICs needs extra cooling.

are no SMDs, it is very easy to solder. Building the car's electronics is not too complicated. The mechanical parts depend highly on the solution you prefer.

The project presented is slightly more than a proof of concept and serves primarily as inspiration for your own projects. **Figure 9** shows my prototype complete with a "roof." The board and software could even form the basis of a completely different device. Nonetheless, there is ample room for improvement. One key area is the steering, which currently behaves too harshly or abruptly. This is because the steering operates like that of a tank: turning left or right involves stopping the respective motor, resulting in nearly right-angled turns. A better approach would be to control the speed of each motor independently, allowing for smoother turns. To achieve this, you would need to modify the program and add one or more sliders to the HTML page — likely implemented with JavaScript.

Overall, the HTML page hosted by the ESP32 could benefit from a more polished design than my version shown in Figure 5. Including information about the distance to the nearest obstacle would also be useful.

230764-01

Questions or Comments?

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

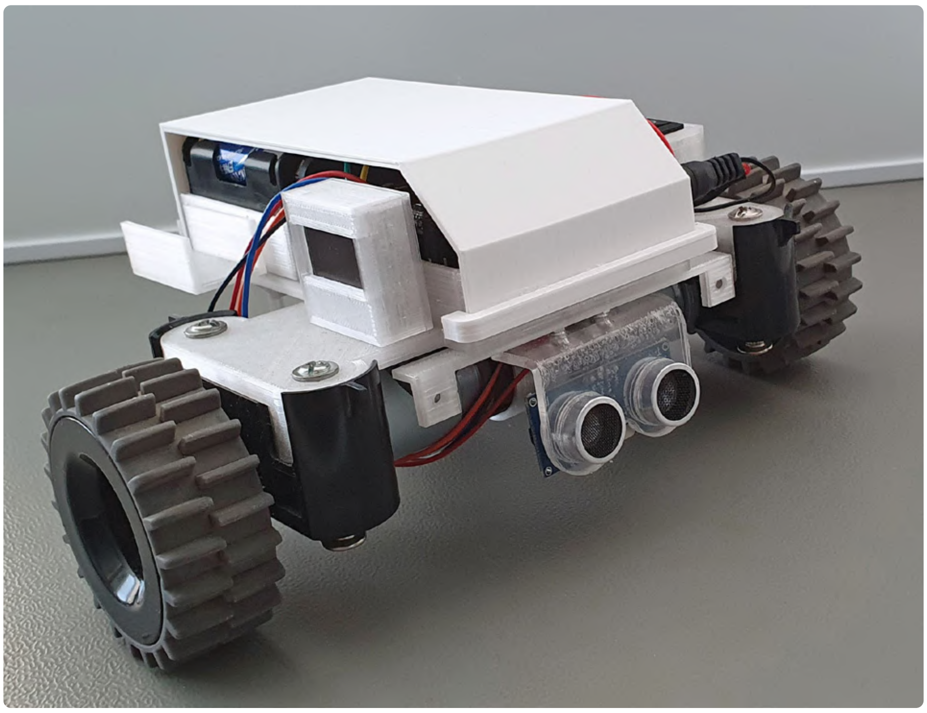


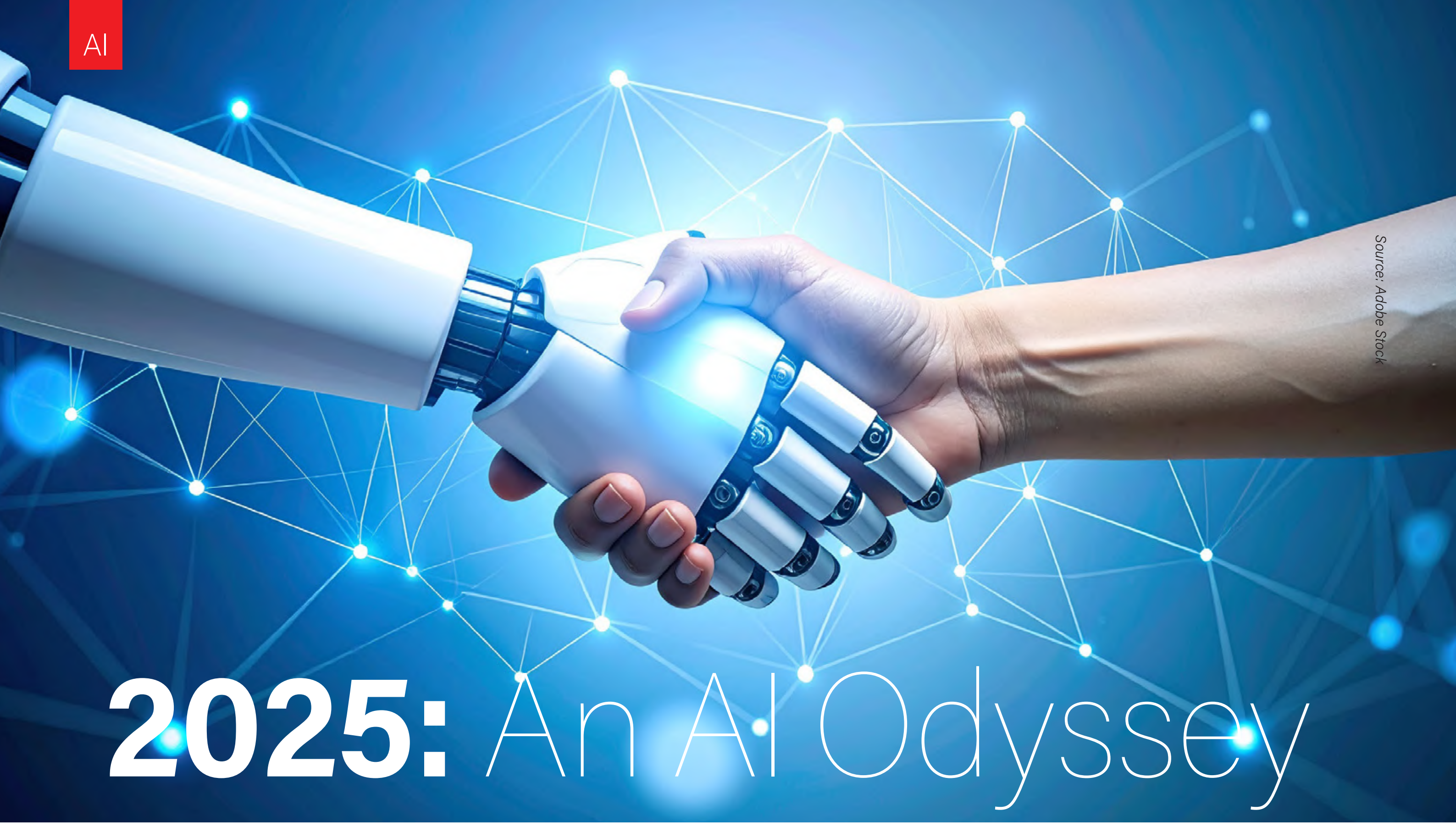
Figure 9: The prototype covered with a white roof appears much more elegant.

About the Author

Huub M. Zegveld studied at the Higher Technical School for Architecture in Eindhoven. Before his retirement he works as a director at a company for work clothing. He likes to deal with electronics, especially Arduino, ESP32, Raspberry Pi, and writing applications with Python. Additionally, he is interested in 3D printing and photography.

WEB LINKS

[1] FreeCAD: https://www.freecad.org
[2] UltiMaker Cura: https://ultimaker.com/software/ultimaker-cura/
[3] Arduino IDE: https://www.arduino.cc/en/software/
[4] Code::Blocks, C/C++ and Fortran IDE: https://www.codeblocks.org
[5] Source and PCB files @ Elektor Labs: https://tinyurl.com/ycyd7p4s
[6] HC-SR04 User Guide, Handson Technology: https://www.handsontec.com/dataspecs/HC-SR04-Ultrasonic.pdf



2025: An AI Odyssey

AI Reasoning Models: The Chain-of-Thought Revolution

By Brian Tristam Williams (Elektor)

Something strange happened in AI, and for once, it wasn't just another model getting bigger and faster. In 2024, we saw a shift from flashy outputs to something deeper: actual step-by-step reasoning. Suddenly, AI started working like engineers do — methodically, logically, and with an internal dialogue instead of a shotgun blast of answers. With the arrival of OpenAI's o3 model in 2025, this new class of "thinking" AI is no longer just a lab curiosity.

Something fundamental changed in AI during 2024, and it wasn't about building bigger models or feeding them more data. The breakthrough came from teaching machines to think step-by-step, the way engineers naturally approach complex problems. OpenAI's o1 introduced this concept [1], but o3's 2025 release [2] has turned systematic reasoning from experimental curiosity into game-changing reality.

For electronics engineers and makers, this represents more than just another AI upgrade. These reasoning models mirror how we actually solve problems — breaking down circuits into functional blocks, systematically isolating faults, and building solutions through methodical analysis rather than intuitive leaps.

The Thinking Revolution

Traditional AI models such as GPT-4 work as sophisticated pattern-matching engines. Feed them a question, and they instantly generate responses based on statistical patterns learned during training. It's an impressive autocomplete, but it's not really thinking.

Reasoning models differ fundamentally. They employ what researchers call a "private chain of thought" — working through intermediate steps before arriving at conclusions. Instead of immediately outputting an answer, o3 actively deliberates, plans ahead, and reasons through tasks much like an experienced engineer approaching a complex design challenge.

Consider how you'd debug a microcontroller communication failure. You don't just guess; you systematically verify power supplies, check signal integrity, examine timing relationships, and test each interface layer. Reasoning models follow this same methodical approach across all problem domains.

Performance Breakthroughs

The benchmark improvements tell a remarkable story. On mathematical reasoning tasks, o3 achieved 96.7% accuracy on the American Invitational Mathematics Examination compared to o1's 74.3 to 83.3%. For software engineering challenges, o3 scored 69.1% on the SWE-Bench Verified Software Engineering benchmark versus o1's 48.9%, representing the difference between occasional help and reliable partnership. [3]

But, the most striking result emerged from the EpochAI Frontier Math benchmark. These problems typically take professional mathematicians hours or days to solve, representing research-level challenges that stump PhD experts. Previous AI models rarely exceeded 2% success rates. o3 solved 25.2% of these problems — a breakthrough that suggests genuine reasoning capability rather than sophisticated memorization.

Another benchmark is the Elo rating. In case you're not a chess nerd or haven't spent weekends buried in Codeforces, Elo rating is a system originally developed to rank chess players [4]. It's since been adapted for competitive programming, where models such as o3 are thrown into the ring with real human coders, solving algorithmic challenges under time pressure. The higher the Elo, the better the performance relative to others. Think of it as a way to gauge not just whether the AI got the right answer, but how skillfully and consistently it did so, in direct comparison with top human talent.

In competitive programming, o3 reached an Elo rating of 2706 to 2727 compared to o1's 1891, placing it among elite human programmers (**Figure 1**). For electronics engineers who rely on algorithmic thinking for embedded software development, this represents a qualitative leap toward genuine programming partnership.

Chain-of-Thought: The Technical Foundation

The breakthrough enabling this reasoning revolution stems from "chain-of-thought" prompting [5]. Research demonstrated that simply asking models to show intermediate reasoning steps — before answering — dramatically improved performance on complex tasks [6].

The mechanism proves elegantly simple: Instead of requesting direct answers, you ask models to work through problems step-by-step. This minor prompt modification fundamentally transforms how models approach challenges, forcing systematic breakdown of complex problems into manageable components.

Chain-of-thought applications benefit electronics work directly [7]. Code debugging becomes systematic analysis rather than trial-and-error guessing. Circuit troubleshooting follows logical diagnostic trees. Component selection weighs multiple constraints simultaneously rather than focusing on single parameters.

Engineering Applications: Beyond Laboratory Demonstrations

For our community, reasoning models enable practical applications that seemed fictional just years ago:

Circuit Analysis Partnership: Describe a power supply specification, and reasoning models walk through complete design processes — calculating component values, analyzing thermal constraints, evaluating efficiency trade-offs, and explaining each decision's rationale. The systematic approach mirrors how experienced engineers actually work.

Intelligent Debugging Assistant: Rather than generic suggestions such as "check connections," reasoning models systematically analyze failure modes. They suggest specific measurement points, explain diagnostic logic, and guide troubleshooting through methodical elimination of potential causes.

Comprehensive Component Selection: Beyond simple parameter matching, reasoning models evaluate environmental

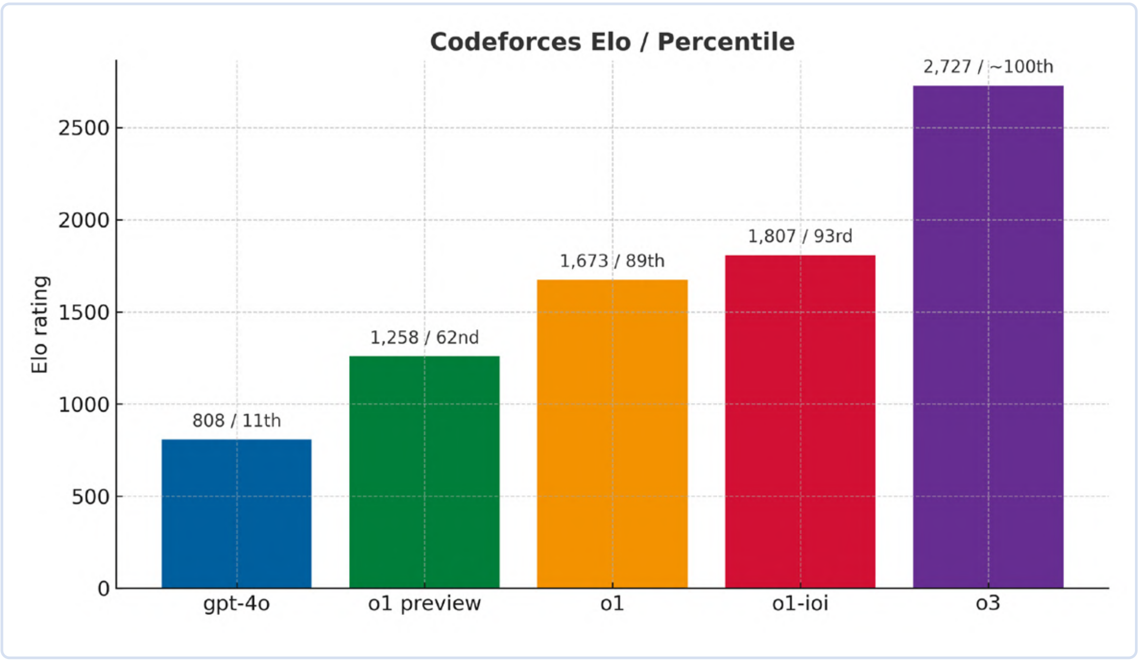
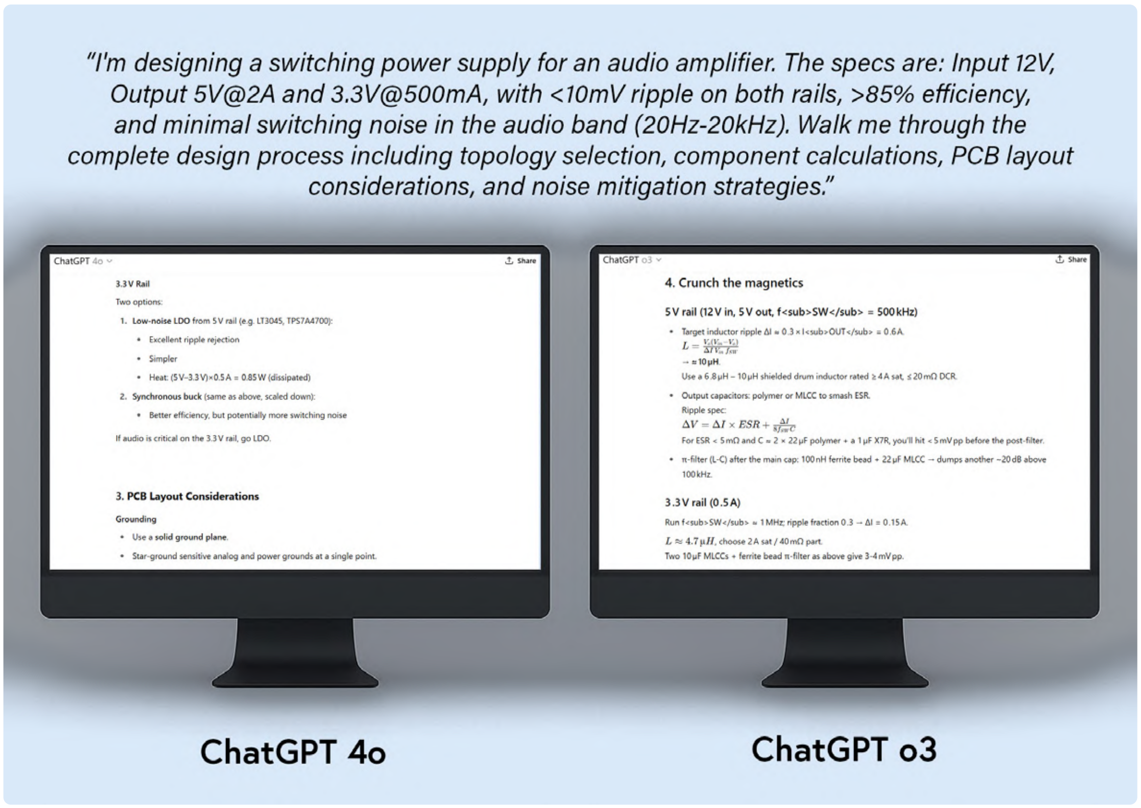


Figure 1: Codeforces Elo ratings and top percentiles for various language models.



"I'm designing a switching power supply for an audio amplifier. The specs are: Input 12V, Output 5V@2A and 3.3V@500mA, with <10mV ripple on both rails, >85% efficiency, and minimal switching noise in the audio band (20Hz-20kHz). Walk me through the complete design process including topology selection, component calculations, PCB layout considerations, and noise mitigation strategies."

ChatGPT 4o

3.3 V Rail

Two options:

1. **Low-noise LDO** from 5 V rail (e.g. LT3045, TPS7A4700):
   - Excellent ripple rejection
   - Simpler
   - Heat: (5 V–3.3 V)×0.5 A = 0.85 W (dissipated)
2. **Synchronous buck** (same as above, scaled down):
   - Better efficiency, but potentially more switching noise

If audio is critical on the 3.3 V rail, go LDO.

3. PCB Layout Considerations

Grounding

- Use a **solid ground plane**.
- Star-ground sensitive analog and power grounds at a single point.

ChatGPT o3

4. Crunch the magnetics

5 V rail (12 V in, 5 V out, f_{SW} = 500 kHz)

- Target inductor ripple ΔI ≈ 0.3 × I_{OUT} = 0.6 A.
  $L = \frac{V_o(V_{in}-V_o)}{\Delta I \, V_{in} \, f_{SW}}$
  → ≈ 10 µH.
  Use a 6.8 µH – 10 µH shielded drum inductor rated ≥ 4 A sat, ≤ 20 mΩ DCR.
- Output capacitors: polymer or MLCC to smash ESR.
  Ripple spec:
  $\Delta V = \Delta I \times ESR + \frac{\Delta I}{8 f_{SW} C}$
  For ESR < 5 mΩ and C ≈ 2 × 22 µF polymer + a 1 µF X7R, you'll hit < 5 mV pp before the post-filter.
- π-filter (L-C) after the main cap: 100 nH ferrite bead + 22 µF MLCC → dumps another ~20 dB above 100 kHz.

3.3 V rail (0.5 A)

Run f_{SW} ≈ 1 MHz; ripple fraction 0.3 → ΔI = 0.15 A.

$L \approx 4.7\,\mu H$, choose 2 A sat / 40 mΩ part.
Two 10 µF MLCCs + ferrite bead π-filter as above give 3-4 mV pp.

Figure 2: ChatGPT 4o versus ChatGPT o3 in complex circuit analysis reasoning.

requirements, analyze cost constraints, consider supply chain factors, and recommend parts based on complete system perspectives rather than isolated specifications.

Firmware Development Partnership: Programming assistance evolves from code completion to genuine collaboration. Models break down complex algorithms into manageable sub-problems, explain implementation trade-offs, and provide detailed rationales for architectural decisions.

Systematic Problem-Solving in Practice

Traditional AI debugging might suggest: "Verify power supply voltages." Reasoning models approach identical problems systematically:

"Let me analyze this step-by-step. The symptoms indicate the device powers on but doesn't respond to inputs, suggesting power delivery works but communication fails. I'll consider the most probable failure points in order of likelihood, starting with signal path integrity, then examining timing relationships, and finally checking protocol-level issues..."

This methodical approach continues through signal analysis, timing verification, and systematic isolation — exactly how experienced engineers attack complex problems, but with unlimited patience and with perfect recall of troubleshooting methodologies.

The contrast between traditional and reasoning AI approaches is stark, with reasoning models providing the systematic methodology that experienced engineers naturally follow.

Three good areas in which to test reasoning skills are:

Complex circuit analysis: I tested both ChatGPT 4o and ChatGPT o3 for the same task: Design a switching power supply. 4o responded almost immediately, while o3 "Thought for 1m 5s." You can see comparative snippets in **Figure 2**, but for the complete outputs, see [8] and [9], respectively. (Just for fun, I asked them for schematics. The two models were equally useless — see **Figures 3** and **4**.)

Multi-variable optimization: Prompt: *"I need to select a microcontroller for a battery-powered IoT sensor that will: measure temperature/humidity every 10 minutes, transmit via LoRaWAN once per hour, operate for 2+ years on 2xAA batteries, cost under $5 in 1000 qty, work from -40°C to +70°C, and fit in a 20mm x 30mm footprint.*

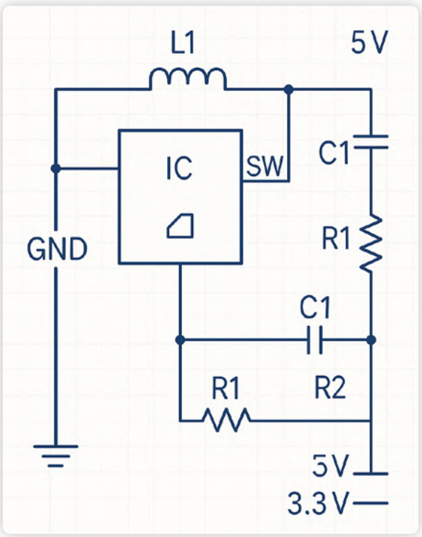


Figure 3: 4o tries to draw me a power supply schematic.

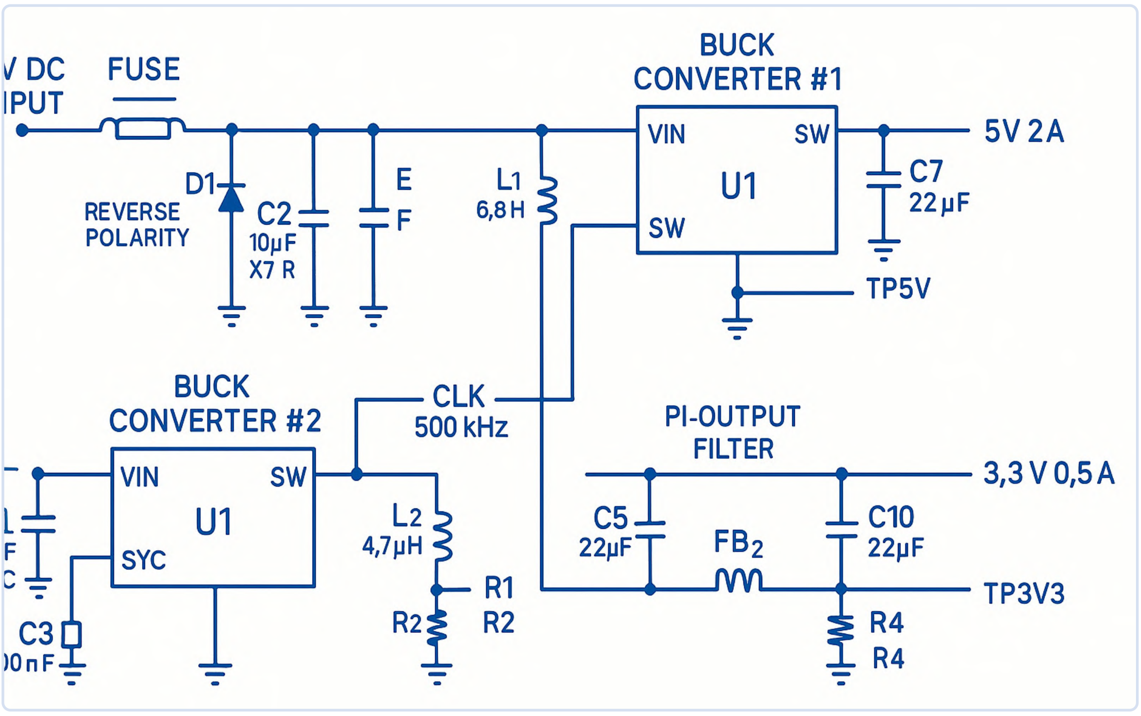


Figure 4: o3 did not fare much better.

Systematically evaluate the trade-offs and recommend the optimal solution with detailed power calculations." This took o3 three minutes and 33 seconds, but the result was impressive [10].

Failure analysis: Prompt: *"My PCB has an intermittent failure where it works fine for hours then suddenly resets. It happens more often when ambient temperature rises above 25°C. The reset line shows clean edges, power rails are stable, and there's no obvious correlation with code execution. Develop a systematic failure analysis methodology to identify the root cause."* Offering a marginal improvement over 4o, o3 deemed this an easy task, thinking for only 23 seconds [11].

Real-World Diagnostic Scenarios

Consider troubleshooting a microcontroller programming failure. Traditional AI suggests checking connections or verifying programmer settings. Reasoning models work through comprehensive diagnostic trees:

Power Verification Stage: "First, confirm target voltage meets specification requirements, checking both steady-state levels and transient response during programming attempts..."

Interface Analysis Phase: "Next, verify programming interface signals — clock edges, data setup times, and reset timing sequences all affect programming reliability..."

Protocol Validation Step: "The programming protocol timing could be critical. Let me examine the specific requirements for your target device and verify tool configuration..."

Environment Assessment: "Finally, consider environmental factors — cable length, ground loops, and electromagnetic interference can affect programming reliability..."

Each stage builds systematically on previous results, with clear explanations of why each check matters and how results guide subsequent analysis. This structured approach separates expert troubleshooters from frustrated beginners.

Educational Revolution

Perhaps most significantly for makers and hobbyists, reasoning models excel at teaching through transparent problem-solving processes. Traditional AI provides answers; reasoning models show their work, transforming opaque responses into educational experiences.

Learning complex topics such as RF design or signal processing becomes interactive collaboration rather than passive consumption. Models adapt explanations to specific knowledge gaps, detecting that you understand basic electronics but haven't worked with feedback control systems, then adjusting explanations accordingly.

Adaptive Technical Instruction

Traditional tutorials follow fixed paths regardless of individual knowledge gaps. Reasoning models customize explanations dynamically. For example, learning switch-mode power supplies, a model might recognize your op-amp experience but your unfamiliarity with compensation networks:

"Since you understand op-amp operation, I'll explain the error amplifier as a specialized comparator configuration. But let me walk through compensation network design systematically — that's where most people encounter difficulties with these topologies..."

This adaptive approach accelerates learning by meeting you precisely where you are, rather than forcing generic explanations that may be too basic or advanced.

Project Planning and Risk Assessment

Reasoning models excel at systematic project planning because they work through potential challenges methodically. In the example of planning an IoT sensor deployment, models reason through:

Technical Constraint Analysis: “Outdoor deployment requires considering temperature ranges, humidity protection, and power consumption optimization. Let me systematically evaluate each constraint and suggest solutions...”

Component Availability Assessment: “Given current supply chain conditions, I’ll suggest alternative components with equivalent functionality, explaining any design modifications required for each substitution...”

Development Timeline Estimation: “Based on RF implementation complexity and your experience level, here’s a realistic development schedule with key milestones and potential risk factors...”

This comprehensive approach prevents common project failures caused by inadequate upfront analysis.

Tool Integration and Capability Enhancement

Reasoning models in 2025 gained tool access — they can search documentation, run simulations, analyze files, and execute code while maintaining systematic reasoning approaches. For electronics work, this enables:

Real-time Datasheet Analysis: Models access component specifications during design discussions, analyzing relevant parameters and explaining implications for specific applications rather than simply retrieving raw data.

Simulation Integration: Models run circuit simulations and interpret results within conversations, explaining why certain behaviors occur and suggesting design modifications based on simulation outcomes.

Current Market Analysis: Models cross-reference component pricing and availability while suggesting alternatives, reasoning through cost-performance trade-offs based on current market conditions.

Code Generation and Testing: Models generate microcontroller code with complete explanations, test functionality, and debug issues through systematic analysis rather than trial-and-error modification.

The reasoning capability ensures tools are used intelligently rather than mechanically. Models don’t just retrieve datasheets, but analyze relevant specifications and explain implications for your specific design challenges.

Programming Partnership Evolution

Software development represents reasoning models’ most mature application area. Rather than generating isolated code snippets, models provide a comprehensive programming partnership:

Systematic Code Analysis: “Examining this interrupt handler, I notice the critical section duration could affect system responsiveness. The nested interrupt structure also presents potential priority inversion risks...”

Explained Code Generation: “I’ll implement this state machine using enumerated types and `switch` statements. This approach provides better maintainability than `if-else` chains for embedded applications because...”

Methodical Debugging: “The symptoms suggest timing-related issues. Let me systematically analyze interrupt priorities, examine task scheduling, and verify peripheral timing requirements to identify probable causes...”

This represents qualitative advancement from code completion to genuine programming collaboration.

Performance Trade-offs and Practical Considerations

Reasoning models deliberately sacrifice speed for thoroughness. Where GPT-4 responds instantly, o3 might require 30 seconds for complex analysis. For engineering applications, this trade-off often makes perfect sense.

Designing power supplies and debugging communication protocols aren’t tasks where instant answers provide value. You want comprehensive analysis, alternative consideration, and clear reasoning behind recommendations. The systematic approach significantly reduces error likelihood compared to quick responses.

Future Integration Prospects

The fundamental shift from *model scaling* to *reasoning time* represents permanent change in AI development direction. Rather than building larger models, researchers focus on giving existing architectures more deliberation time.

This approach offers practical advantages for engineers: Reasoning models can run on existing hardware without requiring massive computational resources. The trade-off remains time versus thoroughness — models think more slowly but much more carefully.

Engineering Software Integration

We can expect reasoning capabilities integrated with traditional engineering tools. Imagine SPICE simulation with AI that reasons through circuit behavior, or PCB layout tools with AI that systematically considers signal integrity, thermal management, and manufacturing constraints.

The key difference will be transparency. Instead of black-box optimization, you’ll observe reasoning behind every suggestion, enabling learning from AI approaches while building genuine expertise.

Embedded Intelligence Possibilities

For embedded systems developers, reasoning models suggest intriguing on-device intelligence possibilities. While current reasoning models require significant computational resources, architectural efficiency trends suggest basic reasoning capabilities could eventually run on microcontrollers.

Consider sensors that reason through fault conditions, or control systems that explain decision-making for safety-critical applications. The systematic approach makes reasoning models particularly suitable for applications requiring explainability.

The Collaborative Future

Importantly, reasoning models transform human-AI interaction from command-response to genuine collaboration. Instead of requesting answers, you engage in technical discussions, explore alternatives, and build understanding together.

This collaborative approach transforms AIs from productivity tools into learning accelerators. Every interaction becomes an opportunity to deepen understanding of underlying principles, not just solve immediate problems.

So, the revolution is about AI that thinks systematically like engineers do. For professionals who live in systematic problem-solving worlds, that represents a genuine game-changer worth serious attention.

We're witnessing the emergence of AI that doesn't just know things, but reasons through problems the way we do. For an engineering community built on methodical analysis and systematic design, this cognitive alignment opens possibilities we're only beginning to explore.

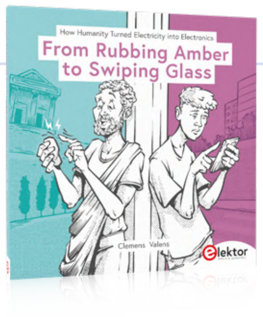
230181-O-01

Questions or Comments?

If you have questions or comments, brian.williams@elektor.com is my email address. You can also catch me on Elektor Engineering Insights each month on YouTube, and you can find me @briantw on X.

About the Author

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


How Humanity Turned Electricity into Electronics
From Rubbing Amber to Swiping Glass
Clemens Valens
elektor

WEB LINKS

[1] Learning to reason with LLMs, OpenAI: https://openai.com/index/learning-to-reason-with-llms/
[2] Introducing OpenAI o3 and o4-mini, OpenAI: https://openai.com/index/introducing-o3-and-o4-mini/
[3] OpenAI's o3: Features, o1 Comparison, Benchmarks & More, datacamp: https://www.datacamp.com/blog/o3-openai
[4] Elo Rating System, Wikipedia: https://en.wikipedia.org/wiki/Elo_rating_system
[5] Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, arXiv.org: https://arxiv.org/abs/2201.11903
[6] Language Models Perform Reasoning via Chain of Thought, Google Research: https://tinyurl.com/chainofthoughtreasoning
[7] What is Chain-of-Thought Prompting?, TechTarget: https://tinyurl.com/chainofthoughtprompting
[8] Complex circuit analysis using ChatGPT 4o: https://tinyurl.com/circuitanalysis4o
[9] Complex circuit analysis using ChatGPT o3: https://tinyurl.com/circuitanalysiso3
[10] Multi-variable optimization using ChatGPT o3: https://tinyurl.com/multivaro3
[11] Failure analysis using ChatGPT o3: https://tinyurl.com/failureo3

Solar Charge Controller with MPP Tracking (3)

Software and Commissioning

By Roland Stiglmayr (Germany)

The first part of this series covered the basics, advantages, and operating principles of an MPP tracker, concluding with a block diagram illustrating the design of an MPPT. Building on this, the second part presented a functional circuit and explained its operation in detail. This third and final part covers the software and demonstrates commissioning.

After covering the basics in the first part [1] of the article series and describing the functioning circuit in the second article [2], the focus now shifts to the software of the MPP tracker and its final commissioning. However, even if you do not plan to build the tracker, the software surely contains interesting routines, such as the ADC routine or the class for controlling the LC display, which are worth a closer look. The program structuring with deterministic execution shows a way to avoid inefficient "spaghetti programming."

The Software

The program *MPPT_6-9* was created in the Arduino programming environment and largely follows Arduino standards. Only the accesses to timer, watchdog, and ADC do not use the libraries, but are performed via direct register access of the ATmega. A custom class was developed for the display, found under *CLASS_DOGS104_IIC*. More details in [3].

The program is extensively commented and contains further explanations. At the beginning of the program listing, values are assigned to constants (`#define`), which are finally adjusted for calibration of the analog part and are used to configure the system. **Listing 1** shows an excerpt from the program.

As usual, in the `Setup` routine, the peripherals such as ports, timers, and ADC are configured, and the external components are initialized. The operating parameters that need to be entered depending on the system are checked for plausibility. As long as the input voltage is insufficient for operation (less than 80% of $V_{PV,OC}$) or the battery voltage is too low, program execution remains in `Setup` and reduces current consumption by activating the *Power Down* mode. If everything is okay, the main program `loop` is called. The main program specifies a fixed cycle time of 4 ms, within which all functions are called. To achieve this, the *Timer2*-interrupt sets the flag `t2_rdy` every 4 ms, which is awaited in `loop` (**Listing 2**), and then a new cycle starts. The function `MPP_Track` also reads TCNT2 to obtain even finer time resolution. The result is deterministic program execution.

The loop() function

As just described, `loop` calls all functions required by the system. It also continuously monitors the input voltage and activates the *Power Down* mode if the voltage falls below the value `VIN_Sdown`. The function `Low_PowerOperation` is called for this, which hands control back to the main program when the input voltage rises again.

In `loop`, the parameters for the MPP search, which depend on the current operating conditions, are also determined. More on this later.

Battery Management

The battery management [4], which controls load switching and the charging process, is fully integrated in the main program. If the battery voltage falls below the value `Load_OffVoltage`, the load is shed to prevent deep discharge of the battery. If there is sufficient sunlight, the battery is charged. If the battery voltage rises above the value `Load_OnVoltage`, the load is reconnected.

As long as sufficient input power is available, the battery is charged. The charging current depends on the input power but is limited to `I_ChargeMax`. When the maximum charging voltage `Max_ChargeVoltage` is reached and the charging current simultaneously drops to 10% of `I_ChargeMax`, it switches to trickle charge. For this, the charging voltage is reduced to 96% of `Max_ChargeVoltage`. If the battery voltage falls below `Low_ChargeVoltage` (corresponds to 93% of `Max_ChargeVoltage`), normal charging is resumed. It must be considered that the load current is superimposed on the charging current. Therefore, the criterion for trickle charge is never reached under load.

Listing 1: Excerpt from the program listing showing the definitions of the configuration parameters.

```
// basic system settings in mV or mA
#define Max_ChargeVoltage 14400        // highest charging voltage of battery
#define Max_ChargeCurrent 15000        // maximum charge current depending on battery
#define Load_OffVoltage   10800        // battery voltage below shuts off the load
#define PV_OpenVoltage    44500        // specified open circuit voltage of solar panel

// valid for Lead-Acid batteries
#define Mx_ChargeVoltage   (Max_ChargeVoltage * 0.99) // max voltage w tolerance
#define Flo_ChargeVoltage  (Max_ChargeVoltage * 0.96) // float voltage
#define Low_ChargeVoltage  (Max_ChargeVoltage * 0.93) // battery voltage below switches to max volt
#define L_ChargeCurrent    (Max_ChargeCurrent * 0.10) // charge current below switches to float volt
#define Load_OnVoltage     (Load_OffVoltage * 1.16)   // battery voltage higher enables load

// scaling of measurements
#define V_Ref      3293       // AREF @ Nano
#define ADC_Off    0.5        // ADC offset in parts of LSB

#define S_Vin      22994      // slope Vin, ratio R-divider * 1024
#define S_VBat     11320      // slope V_Bat, ratio R-divider * 1024
#define S_VO       S_VBat     // slope VO, same as slope V_Bat

#define S_Cur      19163L     // slope Current = (1000mA/55mV) * 1024
#define Ofs_Cur    -110       // current offset/mA, correcting sensor

// scaling of setpoints
#define V_Set0     70                        // voltage offset in mV
#define V_SetZ     long((V_Ref) * 43939L)     // setpoint voltage zero
#define V_SetG     4747L                     // setpoint voltage gain

#define I_Set0     100                       // current 1/mA
#define I_SetZ     (V_Ref * 2048L)            // setpoint current zero
#define I_SetG     ((31744L * 4096) / S_Cur)  // setpoint current gain
```

Listing 2: Query of the flag set by the Timer2 interrupt to determine the cycle time.

```
void loop()
{
    static word ISet_Last;        // last setting of current
    static word Last_Iout;        // last measured current

    digitalWrite(test, LOW);     // signal loop start

    while (!t2_rdy)                // wait until timer flag is set
    {                              // use time to test on low voltage
        ADC_Read(MUX_VIN);               // request VIN
        if (V_Input < VIN_Sdown) {       // test on lowest acceptable VIN
            Low_PowerOperation();        // start low power procedure
        }
    }

    t2_rdy = false;              // now start a new cycle

    digitalWrite(test, HIGH);    // signal calibration start
…
```

A/D Conversion

In `Setup` the ADC is configured for external reference voltage, a clock frequency of 250 kHz, and to use *Polling Mode*. The ADC is called via the function `ADC_Read(A_MUX)`. At the beginning of the function, the multiplexer is set to the passed channel and given 20 µs to stabilize. Then the conversion starts, the result is read out, and converted to millivolts. The conversion is as follows:

```
ADC_Result = ADC_Reading * (V_Ref / 1024);
```

`ADC_Result` is the voltage at the ADC input in mV. `V_Ref` is AREF in mV.

To compensate for any offset error of the ADC, the value `ADC_0` is added. This compensation applies equally to each analog channel (**Listing 3**).

At the end, `ADC_Read` calls the function `Analog_CalcVal(A_MUX, ADC_Result)` (**Listing 4**). From the passed multiplexer address, this function knows which analog value to calculate and where to store the result. For example, to measure the input voltage, `ADC_Read(MUX_VIN)` is called. `ADC_Read` passes `MUX_VIN` and `ADC_Result` to the function `Analog_CalcVal`, which calculates the final result and stores it in the variable `V_Input`.

The slopes for the individual analog values are stored in the definition section of the program. These are determined from the real voltage divider ratios of the attenuators at the analog inputs. To work with fast integer arithmetic and keep rounding errors small, the slopes are multiplied by 1024. Before `Analog_CalcVal` writes the result to the target address, it removes the scaling.

For the example `V_Input (V_I)` with the voltage divider R59 = 100 kΩ and R60 = 4.7 kΩ results in:

ADC_Result = V_I × (R60 / (R59 + R60))
// ADC_Result is voltage at ADC in mV
V_I = ADC_Result × (R59 + R60) / R60)
Slope = S_Vin = (R59 + R60 / R60) × 1024 = 22811
V_Input = (ADC_Result × S_Vin) / 1024

When measuring current `I_Output (I_BCK)`, the slope `S_Cur` is applied in the same way and then the value `Z_Cur`, which represents the zero value, is subtracted. By adding `Ofs_Cur`, the individual zero-point error of the sensor is compensated:

I_Output = ((ADC_Result × S_Cur – Z_Cur) / 1024) + Ofs_Cur
with
Z_Cur = V_Ref × S_Cur/ 2

Calculation of Set Values and Output Via the D/A Converter

In the chapter *Control Circuit* of the second part, the formulas for calculating the control variables for the output current and output voltage of the buck converter were already developed. As a result of these formulas, one gets the two output values of the D/A converter, which are recalculated every 4 ms. To spend as little time as possible on the calculation, integer arithmetic is used. Therefore, the constants used in the calculations are expanded by 2^{16} (see part 2 [2]).

When calling the function `DAC_Write(word V_Val, word I_Val)` (**Listing 5**), the desired output voltage of the buck converter in millivolts is passed to `V_Val` and its output current in milliamperes is passed to `I_Val`.

From the passed values and the constants `V_SetZ`, `V_SetG`, `I_SetZ`, and `I_SetG`, whose values are derived from the formulas, the function calculates the codes for the DAC and passes them to it via SPI.

Listing 3: The ADC_Read function measures the voltage at the selected input in millivolts.

```
void ADC_Read(byte A_MUX)
{
    volatile long ADC_Result;
    ADMUX = ADMUX & 0xf0;               // clear register MUX address
    ADMUX = ADMUX | A_MUX;              // set register to requested MUX address
    delayMicroseconds(20);              // MUX gets stationary
    ADCSRA = ADCSRA | ADC_START;        // now start conversion
    while ((ADCSRA & ADC_START) == ADC_START) {} // wait until conversion is ready
    ADC_Result = word(ADCL);            // read low byte from ADC
    ADC_Result = (ADC_Result + ((ADCH & 0x03) * 256)); // two bits from high byte
    ADC_Result = (ADC_Result * V_Ref) / 1024; // analog voltage at ADC in mV
    ADC_Result = ADC_Result + ADC_0;         // add offset voltage
    Analog_CalcVal(A_MUX, ADC_Result);    // calculate corresponding value, write
                                        // result to target address
}
```

Listing 4: Calculation, scaling, and storage of analog values in Analog_CalcVal.

```
void Analog_CalcVal (byte A_MUX, long ADC_Result)
{
    #define Z_Cur ((V_Ref*S_Cur)/2)                // scaled value of zero current
    volatile long temp;                            // temporary
    if (A_MUX == MUX_ACS)                          // output current
    {
        temp = (ADC_Result * S_Cur) - Z_Cur;       // zero current @ V_Ref/2 * 1024
                                                   // S_Cur= 1000mA/55mV * 1024
        temp = (temp >> 10) + Ofs_Cur;             // remove extension, compensate offset
        if (temp < 0) temp = 0;                    // no negative values
        I_Output = word(temp);                     // result to I_Output
    }
    if (A_MUX == MUX_VIN)                          // input voltage
    {
        temp = (ADC_Result * S_Vin) >> 10;         // S_Vin= ratio R-divider * 1024
        V_Input = temp;                            // V_Input of type long
    }
    if (A_MUX == MUX_VBat)                         // battery voltage
    {
        temp = (ADC_Result * S_VBat) >> 10;        // S_VBat= U_bat * 1024
        V_Bat = word(temp);                        // write battery voltage
    }
    if (A_MUX == MUX_VO)                           // output voltage
    {
        temp = (ADC_Result * S_VO) >> 10;          // S_VO= ratio R-divider * 1024
        V_Output = word(temp);                     // write output voltage
    }
}
```

Listing 5: Calculation of the output codes and writing them to the DAC by using DAC_Write (word V_Val, word I_Val).

```
void DAC_Write(word V_Val, word I_Val)
{
    long temp = long(V_Val) + V_Set0;                  // offset compensation
    temp = V_SetZ - (temp * V_SetG);                   // calc DAC word
    temp = temp >> 15;                                 // clear extension, mul by 2
    if (temp > 4095) temp = 4095;                      // if voltage < min
    if (temp < 35) temp = 35;                          // if voltage > max
    word V_DAC = word(temp);                           // make a word
    V_DAC = (word(V_DAC) | 0x1000 | 0x2000);           // value | enable | gain
    temp = long(I_Val) + I_Set0;                       // compensate offset error
    temp = (temp * I_SetG) + I_SetZ;                   // calc DAC word
    temp = temp >> 15;                                 // clear extension, mul by 2
    if (temp > 4095) temp = 4095;                      // if current > max
    if (temp < 0)    temp = 0;                         // if current < min
    word I_DAC = word(temp);
    I_DAC = (I_DAC | 0x1000 | 0x2000 | 0x8000);        // value or enable or gain or QB

      // now send set voltage and set current to DAC
    digitalWrite(DAC_DIS, LOW);                        // enable DAC
    SPI.transfer16(V_DAC);                             // send voltage
    digitalWrite(DAC_DIS, HIGH);
    digitalWrite(DAC_DIS, LOW);
    SPI.transfer16(I_DAC);                             // send current
    digitalWrite(DAC_DIS, HIGH);
}
```

Listing 6: Excerpt from loop: generation of initial values for MPP_Track.

```
// generate temperature compensated V_MPP from PVoc
if (charge_start && charge_en && !charge)   // test process cycle
{
    if ((T2_CycleCounter & 0x000f) == 0x08) // t = 32 ms, take V_InZero
    {
        V_InZero  = V_Input;                // OC voltage @ low current
        V_InLast  = V_Input;                // set last voltage
        V_MPP        = V_InZero / 2 + V_InZero / 4;  // start with 75% from Voc
        V_MPPstep    = V_MPP / 64;                  // MPP voltage step 1, 6% from Voc
        V_MPPlowest  = V_MPP - V_MPPstep;           // lowest accepted V_MPP @ first run
        V_Ud         = 0;                           // delta voltage
        rise_Cnt     = 0;
        fall_Cnt     = 0;                           // init gradient counter
        first_track  = true;                        // accelerated ramp @ first track
        VIN_Sdown = V_InZero / 2;                   // shut down @ 50%
        if (VIN_Sdown < VIN_MIN)
            VIN_Sdown = VIN_MIN;                    // shut down not < VIN_MIN
        charge = true;                              // now start MPPT charging
        T2_CycleCounter = 0;                        // start timer for new

    // prepare current for start
    if (cal)                                        // if calibration
        Set_IOutput = ISet_Last - ISet_Last / 4;    // continue with 75% of last Set_IOutput
  ...}
...}
```

The MPPT Algorithm

The applied algorithm corresponds to the voltage method as already described in the first part. When searching for the MPP, the differential quotient is also evaluated to speed up the search.

The actual search for the MPP and the subsequent regulation to it is implemented in the procedure `MPP_Track()`. The preparation of the start values for `MPP_Track()` is performed when a recalibration is requested in the main program `loop`. The request is triggered cyclically by `MPP_Track()` itself or after a restart by the *Power Down* mode.

At the beginning of a new search process, the output current is set to 0 and then the input voltage is measured. It corresponds to the open-circuit voltage `Vin_Zero` (=$V_{PV,OC}$) of the solar module. From the open-circuit voltage, the smallest meaningful value of the MPP voltage (`V_MPP`), the absolute lowest MPP voltage (`V_MPPlowest`), the step size `V_MPPstep`, and the power-down threshold (`VIN_Sdown`) are calculated. If the input voltage falls below `VIN_Sdown`, the function `Low_PowerOperation()` is called. The program excerpt in **Listing 6** shows how these values are generated.

On restart (**Figure 1**), the initial current is set to the lowest possible value, and during cyclic recalibration, to 75% of the last measured current value (**Figure 2**). Only now is tracking started via `MPP_Track()` by setting the flag `charge`.

At the beginning of the routine, the step size `I_Stp` is calculated depending on the difference between the input voltage and the predicted MPP voltage (`V_MPP`). This greatly accelerates the search process and thus increases efficiency. The effect can be seen in the different slope of the current ramp in Figure 1. The difference between the current input voltage and the last input voltage is used to calculate the slope `dU`. As long as `V_Input` is greater than `V_MPP`, the current is increased depending on `dU`. Because the `V_MPP` voltage is intentionally chosen small, the MPP is usually passed over, resulting in a sharp drop in `V_Input`. `MPP_Track` responds by increasing `V_MPP` and `V_MPPlowest` by `V_MPPstep` and reducing the last current value by 12.5% (**Listing 7**). This process is repeated until the input voltage no longer falls below `V_MPPlowest` (Figure 2).

The control algorithm of `MPP_Track` keeps `V_Input` at the found value of `V_MPP`. For this, depending on the control deviation (`V_Input` – `V_MPP`) and its sign, the current is adjusted in the smallest steps (`I_Step`). Additionally, the differential quotient dU/dI is formed, and from this the required current change is calculated to quickly respond to larger deviations. Ideally, the regulation lets `V_Input` oscillate around `V_MPP`.

Commissioning the MPPT

First, the assembly is connected to a laboratory power supply without fuse F1 and without Arduino. The power supply's current limit is set to 20 mA, the voltage to 0 V. Now slowly increase the voltage. Up to about 7 V, the current should stay below 0.5 mA, then rise to a value below 10 mA. This is the point where the undervoltage shutdown of IC2 is exceeded. As the input voltage increases further to 10 V, the current drops again. The voltages for the Arduino can be measured at the pads. D_4.8V should be 4.8 V and AREF 3.3 V.

If the input current is higher, there is a fault that must be eliminated. Now one goes back to 0 V and plugs in the Arduino, onto which the

Listing 7: Excerpt from the MPP_Track function: incrementing V_MPP after passing the MPP.

```
if (V_Input < V_MPPlowest) // below lowest
{
    DAC_Write((Set_Voutput), I_Step);         // immediately low current for one cycle
    low_current = true;                       // let I_Start until next cycle

    V_MPP = V_MPP + V_MPPstep;                // increase V_MPP
    V_MPPlowest = V_MPP - 2 * V_MPPstep;      // increase V_MPPlowest

    rise_Cnt = 0; fall_Cnt = 0;               // init gradient counter
    V_Ud = 0;                                 // init gradient generation

    digitalWrite(test0, !digitalRead(test0)); // signal case8

    if (dU < 0)
        Set_IOutput = Set_IOutput - Set_IOutput / 8;   // decrease 12.5%
    else
        Set_IOutput = Set_IOutput - Set_IOutput / 16;  // decrease 6.3%

    first_track = false;                      // finish first tracking

    // upper limit of V_MPP
    if (V_MPP > (V_InZero - (V_InZero >> 4))) // test V_MPP > 94% of Voc
        V_MPP = V_InZero - (V_InZero >> 4);   // set V_MPP to 94%

    if (d) // debug purpose
    {
         Serial.print("8-. ");
        Serial.print(V_MPP); Serial.print(' ');
         Serial.println(V_MPPlowest);
    }
}
```

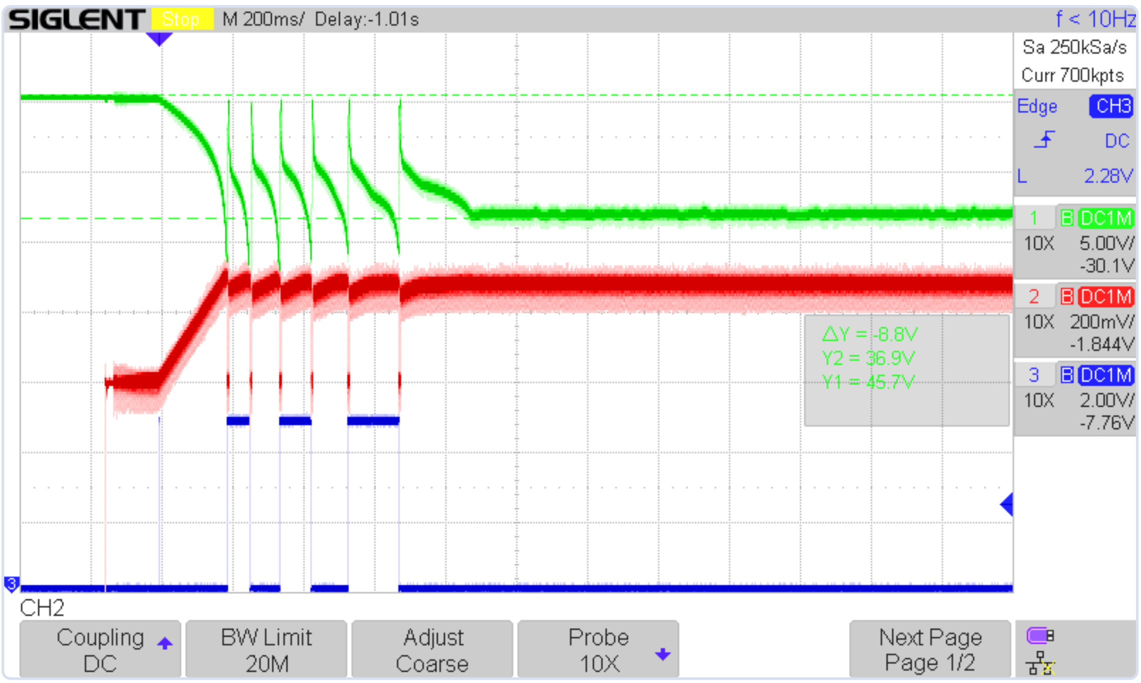


Figure 1: Search process at restart, output current 5 A, V_{OC} = 45.7 V, V_MPP= 36.9 V (green: Input Voltage, red: Output Current, blue: Inc V_MPP).

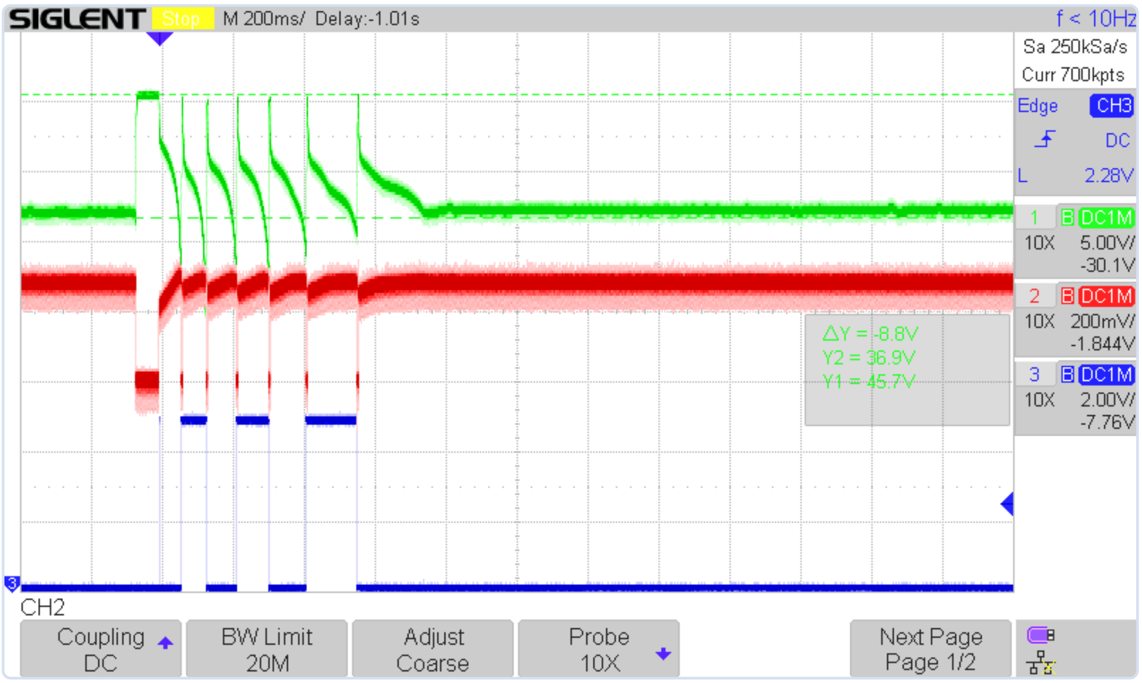


Figure 2: Search during a recalibration, starting with 75% of the last current, output current 5 A (green: Input Voltage, red: Output Current, blue: Inc V_MPP).

program *MPPT_6-9* has been loaded. When increasing the voltage to 8 V, the Arduino should attempt to start. The current must be below 30 mA. After inserting the fuse F1, the Nano should run and the display should show “Low_Vin” and “Low_V_bat.” At 10 V input voltage, the current consumption is below 10 mA. Depending on the intended installation position, the orientation of the text on the display can be adjusted by setting the definition of `LCD_Bot` to `true` or `false`. Next, the Nano is connected to a PC running the Arduino IDE via USB and the program *MPPT_6-9* is loaded.

Since the circuit does not use a potentiometer, some parameters must be set. The definitions in the section `Scaling of Measurements` are provided for this purpose. After every change, the program must be uploaded to the Arduino again.

The accurately measured value of the reference voltage at AREF in mV is assigned to `V_Ref`. In the next step, the battery terminals are shorted, and the resulting voltage Vb is read on the display. If it is higher than 0.02 V, the preset value of `ADC_Off` is reduced by 0.25. If necessary, the value can also be set to 0. The modified program is uploaded to the Arduino.

Then, fuse F2 is removed and 15 V is applied to the battery terminals. If the red LED (LED1) lights up, the voltage at the battery terminals is reversed and must be corrected. After inserting F2, the display should show the correct voltage value. If the displayed value deviates, the slope `S_VBat` is corrected by multiplying it by the factor (Voltage$_{\text{actual}}$ / Voltage$_{\text{display}}$). Since at 15 V the preset value `Load_OffVoltage` for load shedding is exceeded, LED2 lights green.

For further calibration, a simple test program is activated that is operated via the Arduino IDE’s serial monitor. The baud rate is 115200 Bd and inputs are terminated with CR (Carriage Return). For this, set the value `true` in the program line with `#define adjust...` and upload the program to the Nano.

Set the current limit of the laboratory power supply to 150 mA and increase the input voltage to 16 V. As the voltage increases further to about 18 V, the “Low_Vin” message disappears from the display and the 13-V supply is activated. The enable signal from IC10 is also set high by port D9.

The *Test* program is now active, so the output voltage and output current can be set via the serial monitor of the Arduino IDE. Set the output voltage to 5000 mV and the output current to 500 mA. The controller IC4 must now provide a PWM signal that can be measured at the gate-pad of T2. Only now are the two MOSFETs T1 and T2 installed.

After restarting at an input voltage of 30 V, set the current to 4000 mA and the output voltage to 12000 mV in the menu. At the battery terminals, there should now be a voltage of about 12 V. The input voltage is measured and used for calibrating the V_I measurement. The procedure is the same as for the battery voltage; the constant to adjust is `S_Vin`.

When the following mentions “load,” it refers to a load with a load resistor or current sink that allows at least the requested current to flow at the available voltage. In *adjust* operating mode, “output” always means the battery terminal.

Another hint: if a new adjustment is carried out, the offset values `Ofs_Cur`, `V_Set0` and `I_Set0` must first be set to 0.

The output is loaded with about 1 A, and both the real and the displayed current are determined. The measurement is repeated at a load current of about 3 A. From the obtained values, the correction factor for the slope `S_Cur` is calculated.

$$S\_Cur = S\_Cur_{old} \times (I_{actual,3A} - I_{actual,1A}) / (I_{display,3A} - I_{display,1A})$$

After correcting `S_Cur`, the display value at 200 mA load current is read. The difference between the load current and the displayed value is assigned to `Ofs_Cur`. This completes the calibration of the measurements, followed by the calibration of the control variables.

Without a load, set an output of 20,000 mV and 200 mA and measure the actual voltage at the battery terminals. This measurement is repeated at a set output voltage of 10000 mV. From this, calculate the correction factor for `V_SetG` and the offset value `V_Set0`.

$$V\_SetG = V\_SetG_{old} \times (10000 / (U_{actual,20V} - U_{actual,10V}))$$
$$V\_Set0 = U_{actual,20V} - (2 \times U_{actual,10V})$$

Finally, the output current setting is calibrated. Since the software already uses the slope `S_Cur` for current measurement, `I_SetG` is already correct. Only the offset value `I_Set0` is adjusted if necessary. For this, set a current of 200 mA and measure the actual load current. The difference between the setpoint and displayed value is assigned to `I_Set0`.

Now the system is configured, that is, adjusted to the components used. The battery is defined by setting the maximum charging voltage `Max_ChargeVoltage`. All other specific voltage values are calculated from this. The program's default settings refer to a 6-cell lead-acid battery. For other technologies, the settings may need to be adjusted.

Next, the maximum charging current `Max_ChargeCurrent` is defined, which should be about 20% of the battery capacity. Finally, `Load_OffVoltage`, the load shedding voltage at which the load is disconnected from the battery, must be matched.

The characterization of the solar module is done via its open-circuit voltage `PV_OpenVoltage`. As long as the input voltage at startup is less than 80% of `PV_OpenVoltage`, "Low_Vin" is reported.

After setting the constant `adjust` to `false` and selecting the function of the serial interface, the assembly is ready for operation. With `#define plotter false`, all measured values are output as a list, with `true` as plotter string via USB.

250109-C-01

Questions or Comments?

Do you have technical questions or comments about this project? If so, please contact the author (1134-715@online.de) or the Elektor editorial team (editor@elektor.com).

About the Author

Roland Stiglmayr studied information technology in the 1970s and has over 40 years of experience in research and development. His work has focused on the development of computer mainframes, fiber-optic data transmission systems, remote radio heads for mobile communications, and contactless power transmission systems. Today, he works in a consulting capacity, with a particular emphasis on knowledge transfer.

WEB LINKS

[1] Roland Stiglmayr, "Solar Charge Controller with MPPT," Part 1, Elektor 5-6/2025: https://www.elektormagazine.com/250109-01
[2] Roland Stiglmayr, "Solar Charge Controller with MPPT," Part 2, Elektor 7-8/2025: https://www.elektormagazine.com/250109-B-01
[3] Roland Stiglmayr, "Object-Oriented Programming," Elektor 5-6/2021: https://www.elektormagazine.com/200563-01
[4] Roberto Armani, Walter Ribbert, "Pimp My Car Battery Charger," Elektor Circuit Special 2024: https://www.elektormagazine.com/240040-01
[5] Download, Elektor Labs: https://www.elektormagazine.com/labs/solar-charge-controller-with-mpp-tracking

elektorstore

www.elektor.com

Learn FPGA Programming with Verilog

Master FPGA programming with the Red Pitaya Academy Pro Box. Learn Verilog and build a real-time audio processing system using Red Pitaya – with a full online course and hands-on project materials.

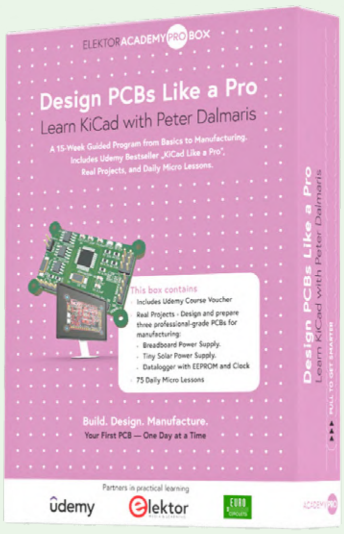
Price: €649.00

Member Price: €584.10

www.elektor.com/21202

Design PCBs Like a Pro

The Academy Pro Box "Design PCBs like a Pro" offers a complete, structured training programme in PCB design, combining online learning with practical application.

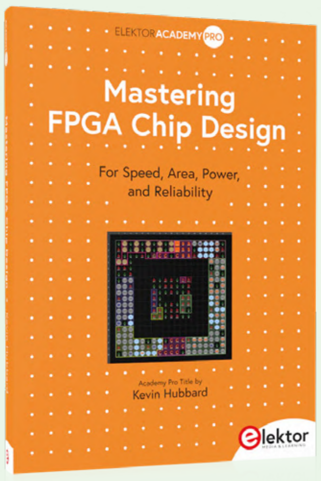


Price: €199.95

Member Price: €179.96

www.elektor.com/21203

Mastering FPGA Chip Design

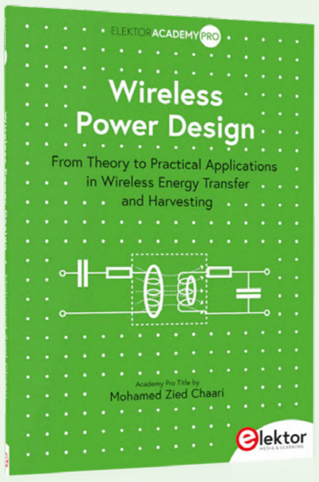


Price: €39.95

Member Price: €35.96

www.elektor.com/21211

Wireless Power Design



Price: €39.95

Member Price: €35.96

www.elektor.com/21177

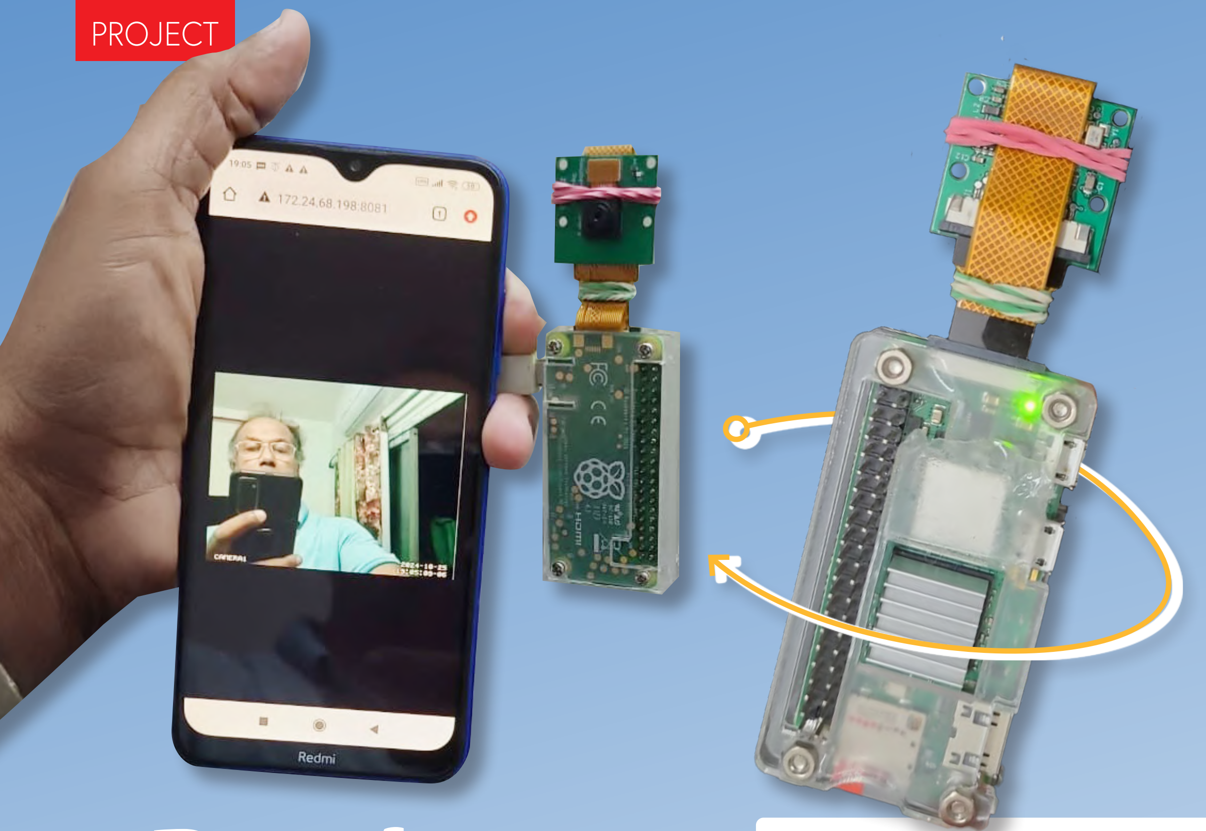


Figure 1: The Raspberry Pi Zero Web Streaming Camera is ideal for long-running streaming applications such as pet monitoring, time-lapse capture, or remote observation.

Raspberry Pi Zero Web Streaming Camera

Using the ZeroTier VPN

By Bera Somnath (India)

The Raspberry Pi Zero Web Streaming Camera is a compact, low-cost DIY project using a Raspberry Pi Zero W, a 5-MP camera module, and a mobile hotspot. Designed for secure, remote video streaming, it uses the Motion program for lightweight webcam functionality and ZeroTier VPN for global access. The project is ideal for surveillance, monitoring, or educational use without relying on third-party cloud services.

Streaming live video from a mobile device or desktop has become almost mundane. Yet, challenges arise when attempting to do the same with ultra-low-cost, low-power hardware, especially when aiming for 24/7 operation that is secure and reliable. That's the motivation behind this IoT project: transforming the small but mighty Raspberry Pi Zero and a 5-MP ZeroCam camera into a globally accessible, one-to-one streaming web camera; using only a mobile hotspot for internet connectivity. Utilizing the Motion program for streaming, the system can also function as a surveillance tool, capturing photos and videos of intruders with a date and time stamp for added security.

The Raspberry Pi Zero is my favorite development board. It's smaller than the Arduino UNO and Maixduino, and it is slightly bigger than the Raspberry Pi Pico, but its footprint is almost the same as the ESP32. However, in terms of capability, the Raspberry Pi Zero is far ahead of these other boards. Only the Orange Pi Zero or NanoPi Neo can come closer in terms of performance. Unlike microcontrollers such as the ESP32 or Raspberry Pi Pico, the Raspberry Pi Zero is a fully-fledged computer, capable of running a complete Linux operating system and performing tasks typically reserved for larger computers, but on a much smaller scale.

Initially born out of curiosity, this project evolved from a simple Raspberry Pi-based photo booth into a highly practical, standalone streaming device. While smartphones already do this with ease, the Raspberry Pi Zero offers a significantly smaller and cheaper alternative, costing only about €15 to €20. Building this is also an excellent learning project, opening the door to a deeper understanding of embedded systems, Linux networking, and practical IoT applications.

Practical Advantages

While many users would naturally reach for their smartphones to stream video, this Raspberry Pi Zero webcam setup offers several compelling advantages. Firstly, it is designed to operate as a standalone dedicated device equipped for continuous operation, which you wouldn't use a smartphone for. This solution works not only over mobile hotspots but can also connect to standard Wi-Fi networks.

Lastly, in terms of privacy and security, the project provides local control without dependence on third-party cloud services. With ZeroTier VPN, remote access becomes seamless and secure, eliminating the need for router configuration or port forwarding. The compact size of the Pi Zero makes it ideal for installations in tight spaces, and its flexibility means it can easily be expanded to include AI features, sensor integration, or automated alerts. Additionally, the setup supports one-to-one video streaming, ensuring efficient bandwidth usage without the overhead of traditional live-streaming platforms.

Ethical Considerations

Given its small size, this setup could be used to record people without their knowledge. It is important to remember that technology itself is neutral; its impact is determined by the intent and integrity of the user. When approached with a sense of curiosity and responsibility, even the smallest DIY project can lead to the most transformative learning experiences.

This project is intended for use cases such as DIY home monitoring, educational exchanges, and personal security. Adding LED indicators, clear signage, and using it only in areas where you have ownership or permission are all part of best practice.

Recording people without their consent is both illegal and unethical in most parts of the world. If anyone considers using such devices covertly, my advice is clear: don't. The right application lies in transparency, consent, and constructive intent.

What You'll Need To Get Started

The system consists of a Raspberry Pi Zero W board [1] connected to a 5-megapixel ZeroCam Raspberry Pi camera module [2] (**Figure 1**). The Pi Zero is connected to the internet via a mobile phone, which acts as a Wi-Fi hotspot. This compact and portable setup allows the Pi Zero to function as a web camera or a motion detection system.

These are the hardware components of the project:

- Raspberry Pi Zero W [1]
- ZeroCam 5 MP camera [2]

All that's required is to connect the camera to the Pi Zero port and power supply and you're good to go.

Software Components

I chose **Motion** [3] as software for the Pi Zero, not necessarily for motion detection, but more because it functions excellently as a lightweight webcam server. Most other options were too resource-heavy for the Pi Zero and either lagged or crashed frequently. Motion was the only one that worked smoothly within this limited RAM and CPU budget. Motion captures video streams from the camera and processes these streams in real-time. When the system detects motion, it can trigger actions such as recording video, taking snapshots, and saving them locally or transmitting them over the network.

ZeroTier [4] is a virtual networking tool that enables secure, encrypted one-to-one communication across the internet without complex networking configurations like port forwarding (**Figure 2**). Once you have understood its philosophy, it is straightforward to use.

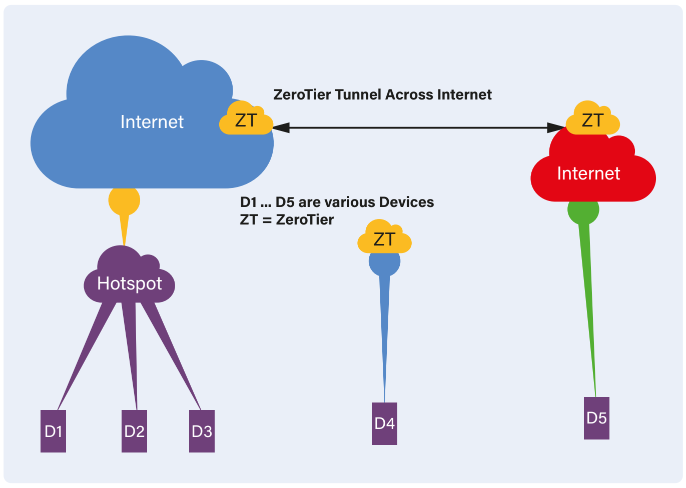

Internet
ZT
ZeroTier Tunnel Across Internet
ZT
Internet
D1 ... D5 are various Devices
ZT = ZeroTier
ZT
Hotspot
D1
D2
D3
D4
D5

Figure 2: ZeroTier creates a virtual tunnel through which your devices can be connected to each other securely.

How To Set Up Motion

Basic essential commands:

`$ sudo apt install motion -y` *(One-time command. This will install Motion.)*
`$ sudo systemctl start motion` *(This will start Motion.)*
`$ sudo systemctl enable motion` *(One time command. This will start Motion at boot level, every time the Pi Zero starts/reboots.)*

The following commands are optional:

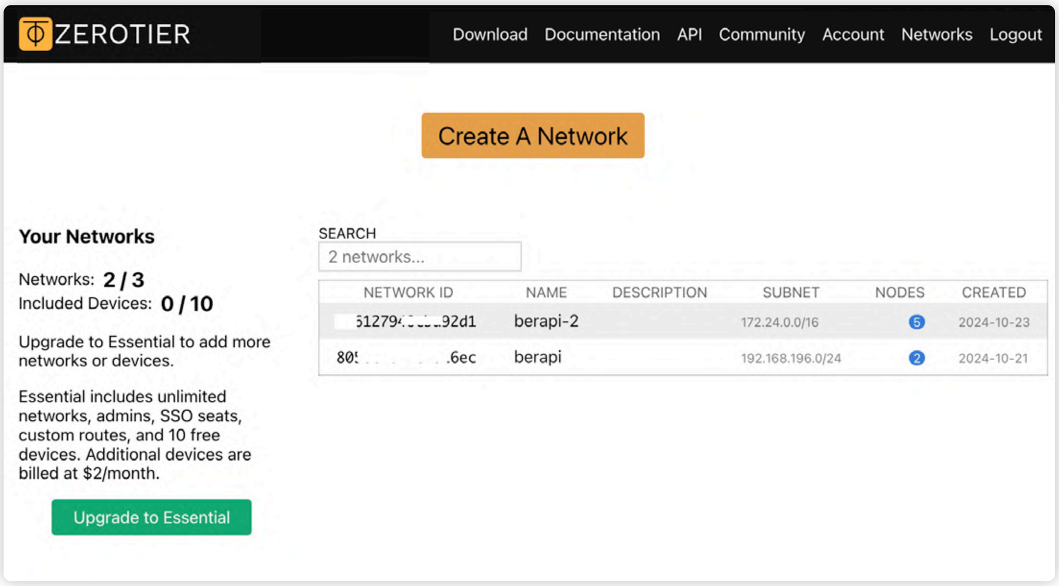
`$ sudo systemctl status motion` *(To check whether Motion is running.)*
`$ sudo systemctl stop motion`
`$ sudo systemctl restart motion`

The config file */etc/motion/motion.conf* can be found on Elektor Labs [5]. Read the file carefully to understand each option, such as how to change the destination of the log file, locating the port on which the streaming will be available etc. Most defaults work out of the box, but advanced users can configure resolution, framerate, streaming port, log directory, and output format.

With Motion software running on your Raspberry Pi Zero, the stream is accessible on `port 8081` within your local network. Any device on the same WiFi network can view it by entering `http://`*<pi-zero-ip>*`:8081` in a browser.

Setting Up Remote Access With ZeroTier

To make the stream available outside your local network (beyond your mobile hotspot's NAT firewall), you will need ZeroTier to create a secure pipeline between your devices.


ZEROTIER

Download Documentation API Community Account Networks Logout

Create A Network

Your Networks

Networks: **2 / 3**
Included Devices: **0 / 10**

Upgrade to Essential to add more networks or devices.

Essential includes unlimited networks, admins, SSO seats, custom routes, and 10 free devices. Additional devices are billed at $2/month.

Upgrade to Essential

SEARCH
2 networks...

NETWORK ID	NAME	DESCRIPTION	SUBNET	NODES	CREATED
[redacted]512794[redacted]92d1	berapi-2		172.24.0.0/16	5	2024-10-23
805[redacted].6ec	berapi		192.168.196.0/24	2	2024-10-21

Figure 3: Select "Create a Network" on ZeroTier Central for your unique Network ID.

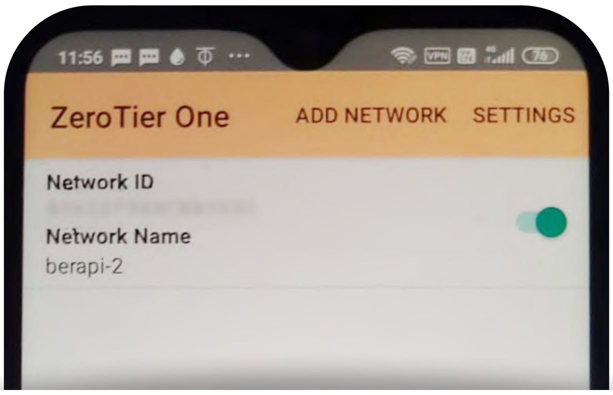
1. **Connect to the internet**
Ensure that your Pi Zero and any devices from which you want to access it are connected to the internet. For this setup, a mobile Wi-Fi hotspot can be used.

2. **Join a ZeroTier network**
Go to ZeroTier Central [4], create an account and log in.

3. **Set up a network**
Click on *Create a Network*, which will generate a unique Network ID (a long hexadecimal string) for your virtual network (**Figure 3**).

4. **Add and authorize devices on ZeroTier**
On each device you want to use, install the ZeroTier One and enter your Network ID. On mobile devices, use the *ZeroTier One* mobile app [6] and connect by entering the Network ID in the *Join Network* section (**Figure 4**).

On Pi Zero, install ZeroTier with this command:


11:56

ZeroTier One ADD NETWORK SETTINGS

Network ID
[redacted]

Network Name
berapi-2

Figure 4: The ZeroTier One app will enable you to connect mobile devices to the network.

```
$ sudo apt install -y zerotier-one
```

Here is the command to connect to the ZeroTier network:

```
$ sudo zerotier-cli join <Network-ID>
```

The Pi Zero has now joined the network, but hasn't yet received its IP address, to be accessible from outside. For this, you first have to go to ZeroTier Central and approve the Pi Zero. In ZeroTier Central, go to *Network settings* and check the authorization box next to the device's name (**Figure 5**). Once authorized, the Pi Zero will get a unique ZeroTier IP address, visible in ZeroTier Central.

Once approved, you may also run the following commands on the Pi Zero to get it's external IP address or get info about the network:

```
$ sudo zerotier-cli listnetworks
$ sudo zerotier-cli info
```

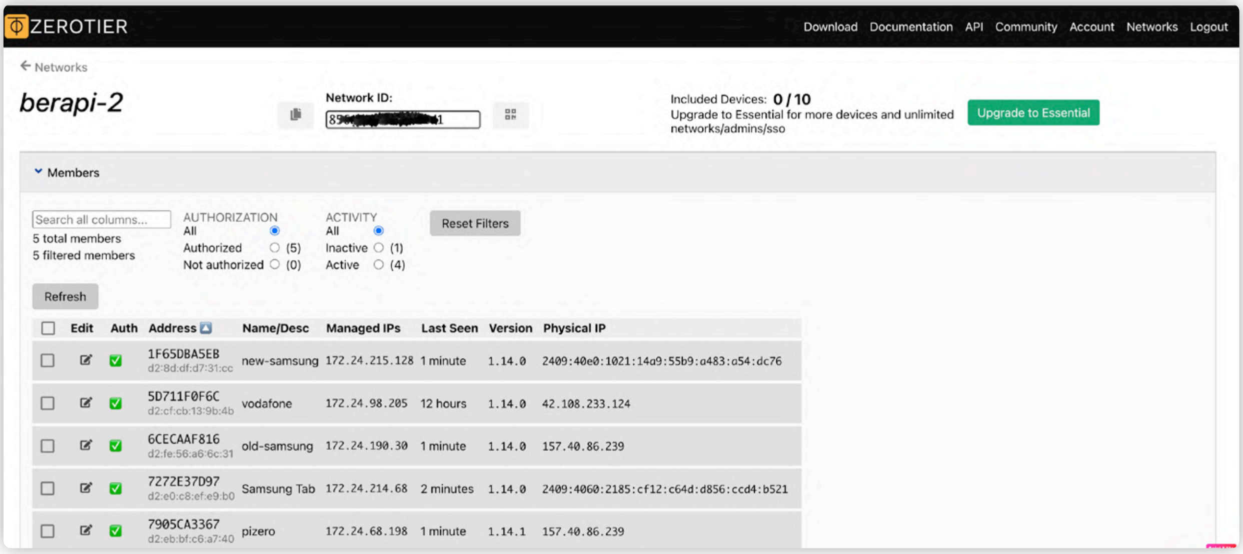
5. **Verify the connection**
Use the Pi Zero's command line to ping other devices on the ZeroTier network:

```
$ ping <ZeroTier-IP-of-other-device>
```

Devices with successful responses indicate an established secure connection. This "invisible pipeline" is a secure VPN tunnel, keeping the stream private and accessible only by authorized devices.

Access Your Stream from Anywhere

Now, open a browser on any authorized device and enter `http://`*<ZeroTier-IP-of-Pi-Zero>*`:8081` to access your video stream. The stream will appear seamlessly, accessible across the world as long as both devices remain connected to the internet.



ZEROTIER

berapi-2

Network ID: [redacted]

Included Devices: **0 / 10**
Upgrade to Essential for more devices and unlimited networks/admins/sso

Search all columns...
5 total members
5 filtered members

AUTHORIZATION		ACTIVITY	
All	●	All	●
Authorized	○ (5)	Inactive	○ (1)
Not authorized	○ (0)	Active	○ (4)

☐	Edit	Auth	Address	Name/Desc	Managed IPs	Last Seen	Version	Physical IP
☐	✎	✓	1F65DBA5EB d2:8d:df:d7:31:cc	new-samsung	172.24.215.128	1 minute	1.14.0	2409:40e0:1021:14a9:55b9:a483:a54:dc76
☐	✎	✓	5D711F0F6C d2:cf:cb:13:9b:4b	vodafone	172.24.98.205	12 hours	1.14.0	42.108.233.124
☐	✎	✓	6CECAAF816 d2:fe:56:a6:6c:31	old-samsung	172.24.190.30	1 minute	1.14.0	157.40.86.239
☐	✎	✓	7272E37D97 d2:e0:c8:ef:e9:b0	Samsung Tab	172.24.214.68	2 minutes	1.14.0	2409:4060:2185:cf12:c64d:d856:ccd4:b521
☐	✎	✓	7905CA3367 d2:eb:bf:c6:a7:40	pizero	172.24.68.198	1 minute	1.14.1	157.40.86.239

Figure 5: Once your devices are connected, you will need to authorize them from your ZeroTier Central dashboard.

Automating VPN Connection on Boot

To ensure the Pi Zero automatically checks and connects to the ZeroTier network every time it boots, let's create a small shell script and set it up to run as a system service. Open a new file to write the ZeroTier connection script:

```
$ nano zerotier.sh
```

You can download a template for this file at [5].

Now, create a systemd service file to run the script at boot.

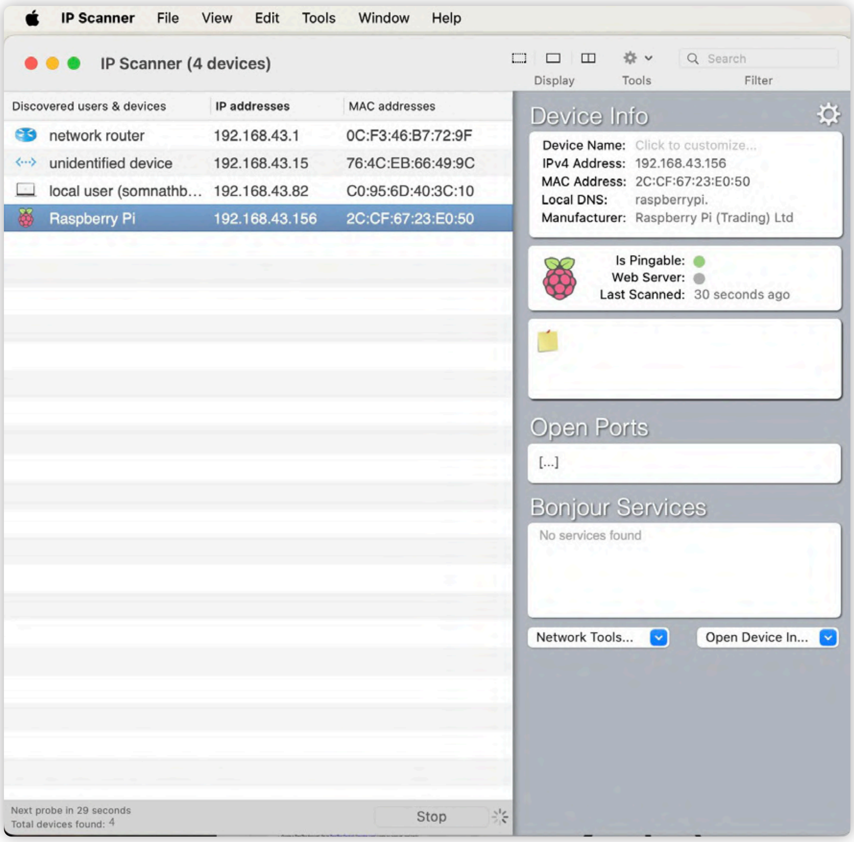
```
$ sudo nano /etc/systemd/system/zerotier.
service
```

You can download a template for this file at [5].

To activate and test the service, run the following commands once:

```
$ sudo systemctl daemon-reload
```
(Reloads systemd to recognize the new service.)
```
$ sudo systemctl enable zerotier.service
```
(Enables the service to run at boot.)
```
$ sudo systemctl start zerotier.service
```
(Starts the service immediately.)
```
$ sudo systemctl status zerotier.service
```
(Optional: checks the service status.)

One tip: Most modern mobile devices, when set up as Wi-Fi hotspots, will no longer display the IP addresses assigned to connected devices. This leads to challenges when trying to identify the correct IP, particularly for tasks like SSH access on a Raspberry Pi Zero. To streamline this process, I recommend installing network-scanning tools such as LanScan [7] or IP Scanner [8] on your laptop or desktop (**Figure 6**).



IP Scanner (4 devices)

Discovered users & devices	IP addresses	MAC addresses
network router	192.168.43.1	0C:F3:46:B7:72:9F
unidentified device	192.168.43.15	76:4C:EB:66:49:9C
local user (somnathb…	192.168.43.82	C0:95:6D:40:3C:10
Raspberry Pi	192.168.43.156	2C:CF:67:23:E0:50

Device Info

Device Name: Click to customize...
IPv4 Address: 192.168.43.156
MAC Address: 2C:CF:67:23:E0:50
Local DNS: raspberrypi.
Manufacturer: Raspberry Pi (Trading) Ltd

Is Pingable: ●
Web Server: ●
Last Scanned: 30 seconds ago

Open Ports

[...]

Bonjour Services

No services found

Next probe in 29 seconds
Total devices found: 4

Figure 6: Apps such as IP Scanner will allow you to identify the correct IP address for your connected devices to simplify remote access and management.

Looking Ahead

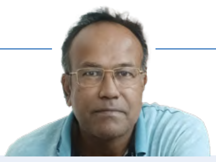
This project demonstrates the versatility of the Raspberry Pi Zero as a compact, low-power solution for live streaming, remote monitoring, and IoT applications – all achieved without the need for high-power, resource-intensive hardware. With its efficient energy consumption, it operates reliably over long durations, even in mobile or battery-powered setups, setting it apart from full-sized Raspberry Pi computers. By leveraging open-source software like Motion for streaming and ZeroTier for secure, remote access, this project offers a stable, self-contained solution for one-to-one streaming and flexible monitoring across vast distances, free from proprietary services or costly subscriptions.

Looking ahead to possible future modifications, the low-power Raspberry Pi Zero can evolve into an even more powerful tool, supporting advanced AI-based image analysis, real-time alerts, and edge computing capabilities. This opens doors for use cases in security, environmental monitoring, and remote data acquisition, making it a future-ready platform for adaptable, real-time applications.

240739-01

Questions or Comments?

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

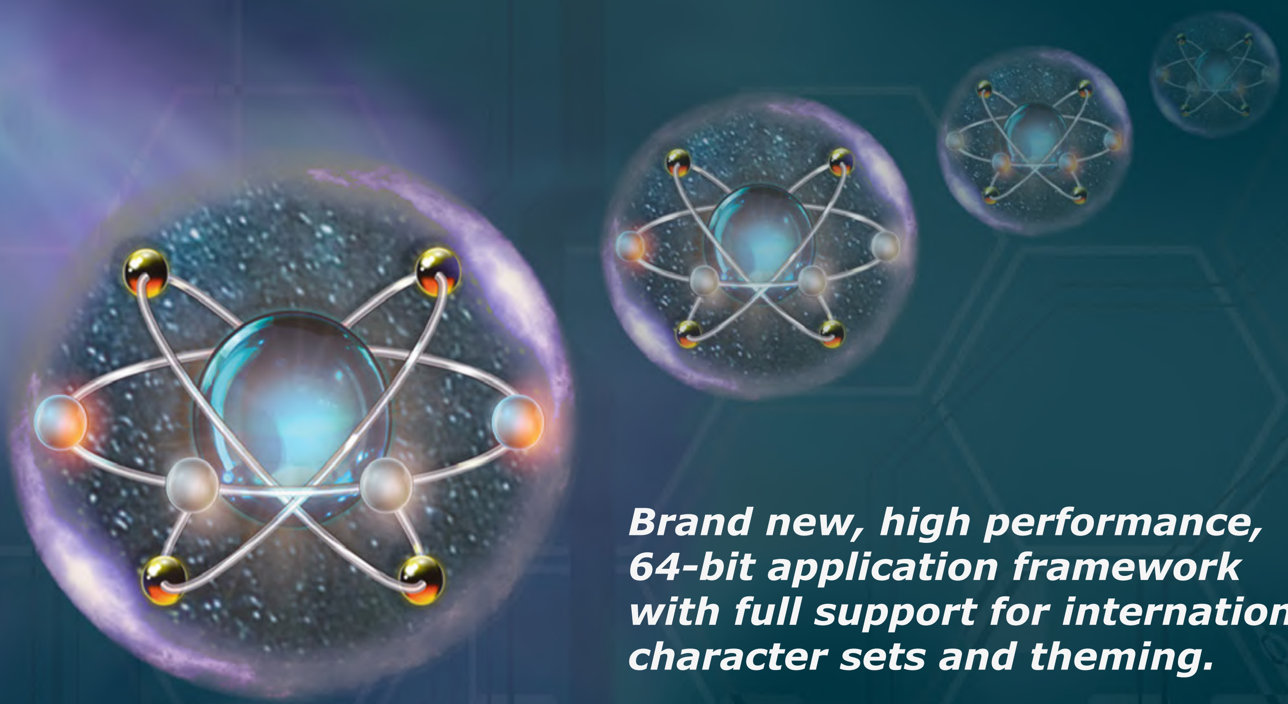


About the Author

Somnath Bera, a mechanical engineer from Jalpaiguri Govt. Engg. College, India, worked for over 36 years at different capacity at NTPC, the largest power producer in the country. He has a profound passion for electronics, evidenced by his 60+ innovative projects on Elektor Labs, over 14 of which have been featured in Elektor. His projects are often focused on problem-solving in areas like waste and natural resource management. Somnath likes to use innovative approaches and platforms like Arduino, Raspberry Pi, and ESP32 coupled with various kinds of sensors and wireless systems to create efficient and cost-effective solutions.

WEB LINKS

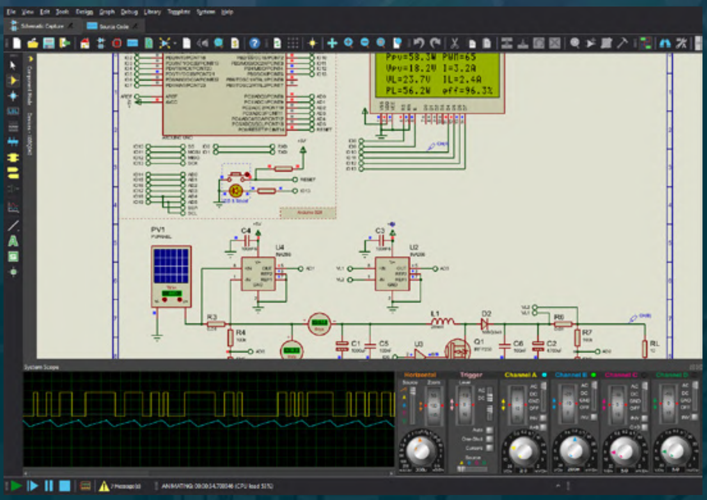
[1] Raspberry Pi Zero, Elektor Shop: https://www.elektor.com/products/raspberry-pi-zero-w
[2] ZeroCam - Camera for Raspberry Pi Zero, The PiHut: https://thepihut.com/products/zerocam-camera-for-raspberry-pi-zero
[3] Motion Software: https://motion-project.github.io/
[4] ZeroTier: https://www.zerotier.com/
[5] Elektor Labs webpage of this project:
https://www.elektormagazine.com/labs/pi-zero-world-wide-web-streaming-on-mobile-hotspot-using-vpn
[6] ZeroTier One, App Store: https://apps.apple.com/us/app/zerotier-one/id1084101492
[7] LanScan, App Store: https://apps.apple.com/us/app/lanscan/id472226235?mt=12
[8] IP Scanner, App Store: https://apps.apple.com/us/app/ip-scanner/id6443819966

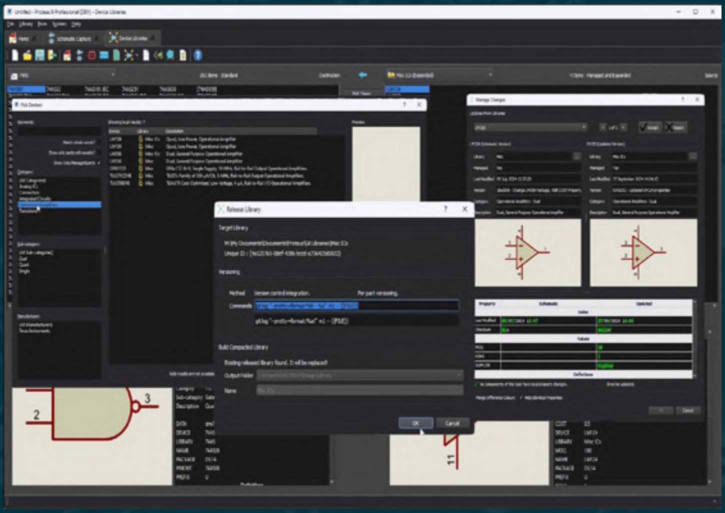


Brand new, high performance, 64-bit application framework with full support for international character sets and theming.

PROTEUS 9

...THE START OF A NEW ERA





Proteus VSM Simulation

- System Scope with live circuit probing.
- Live results on graph based simulations.
- Visual Designer for MicroPython.
- IoT Builder for MicroPython.

Proteus PCB Design

- Versioning of managed libraries with either repository or local folder.
- Software wide support for restricting use to managed libraries.
- Lightning fast, live power plane regeneration.
- Via Inspector tool for board analysis.

labcenter Electronics

www.labcenter.com

Tel: +44 (0)1756 753440

Order - with - **Confidence**

MOUSER ELECTRONICS