£9.95 | €10.95 | $12.95

elektor MAG 503

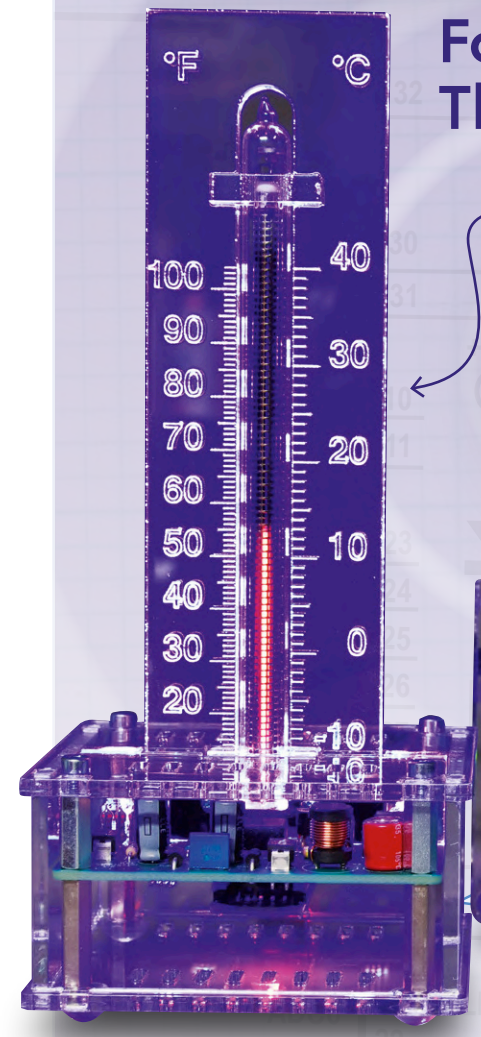
design › share › sell

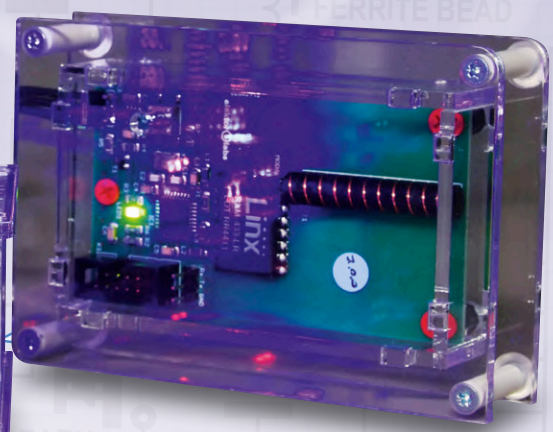
EDITION 5/2020
SEPTEMBER & OCTOBER 2020
ELEKTORMAGAZINE.COM

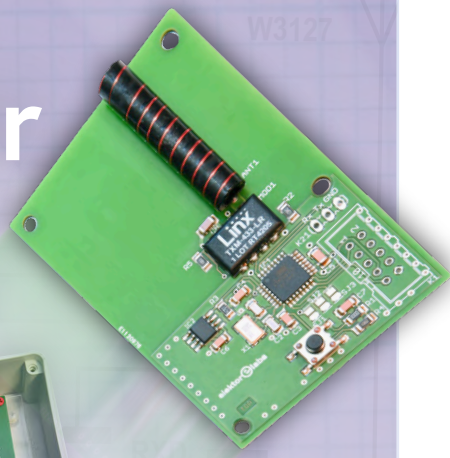
Wireless Temperature Sensor

For the Nixie Bargraph Thermometer

16





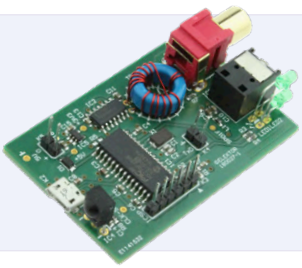


In this edition

- Timer for Headphone Amplifier
- Analogue Filter Theory
- ESP32 Multitasking: Semaphores
- PIC Programming
- Review: Bluetooth Multimeter
- AI for Beginners
- Multitasking with Raspberry Pi
- Draw Circuit Diagrams
- Error Analysis

and much more!

Home Automation Made Easy
With ESPHome, Home Assistant & MySensors
36



USB to S/PDIF Interface
Digital audio output for computer, laptop, tablet or smartphone
6

High-Voltage Power Supply
With Curve Tracer
92

9 770268 451173 00503

Arduino boards are easy to use and inexpensive. With the Elektor AVR Playground Bundle, you can familiarize yourself with the world of Arduino while you learn to program microcontrollers.

AVR Playground Bundle

elektor

Description:

Arduino boards are easy to use and inexpensive. With the Elektor AVR Playground Bundle, you can familiarize yourself with the world of Arduino while you learn to program microcontrollers. It is a handy reference for electronics beginners, students, and even teachers looking for ideas.

The informative bundle includes Mastering Microcontrollers Helped by Arduino (by Clemens Valens, Elektor) and the Elektor AVR Playground Board, which makes it easy to learn about the AVR microcontroller without having to solder or tocomponents. In the book, Valens covers microcontroller basics, including inputs and outputs (analog and digital), interrupts, communication buses (RS-232, SPI, I²C, 1-wire, SMBus, etc.), timers, and much more. With this bundle, theory is put into practice on an Arduino board using the Arduino programming environment.

Welcome to the world of AVR!

Learn more:

www.elektor.com/elektor-avr-playground-bundle-en

COLOPHON

Elektor Magazine,
English edition

Edition 5/2020
Volume 46, No. 503
September & October 2020

ISSN 1757-0875 (UK / US / ROW distribution)

www.elektor.com
www.elektormagazine.com

Elektor Magazine, English edition
is published 6 times a year by

Elektor International Media
78 York Street
London W1H 1DP
United Kingdom
Phone: (+44) (0)20 7692 8344

Head Office:
Elektor International Media b.v.
PO Box 11
NL-6114-ZG Susteren
The Netherlands
Phone: (+31) 46 4389444

Memberships:
Please use London address
E-mail: service@elektor.com
www.elektor.com/memberships

Advertising & Sponsoring:
Margriet Debeij
Phone: +49 170 5505 396
E-mail: margriet.debeij@elektor.com

www.elektor.com/advertising
Advertising rates and terms available on request.

Jens Nickel

International Editor-in-Chief, Elektor Magazine

Carefully **blended**

Elektor is not only appreciated by its readers worldwide for its mixture of theory and practice. We also have a great blend of in-depth articles and content for beginners. From the many reader e-mails we receive, as well as direct conversations, I know that these articles are not only read by 'beginners'. Advanced readers also use them to refresh or round-out their knowledge. In this issue, you will find our popular small circuits series "Starting Out in Electronics" as well as an article explaining how, as a newcomer to electronics, you can create your first computer-drawn circuit diagrams. The step-by-step instructions for the simple CAD program EasyEDA are taken from our special "Getting started in electronics with Arduino" book by Florian Schäffer. It will also be available in English shortly.

You probably already noticed that our magazine has also got a new layout. Our sister magazine Elektor Industry, our Elektor books, and the websites have also received a refresher. What do you think of the new look? Our new Art Director Harmen Heida and I would be happy if you would send us feedback by mail!

I also have some good news for all electronics professionals, especially those who are interested in the very latest developments, both professionally and privately. If nothing completely unforeseen should occur, then the electronica trade fair will open its doors on the 10th of November. This means that the Fast Forward Award, where the best electronics start-ups of the year compete against each other, will also be able to take place as planned (www.elektormagazine.com/fastforward). We also look forward to welcoming many of our readers there as well!

The Team

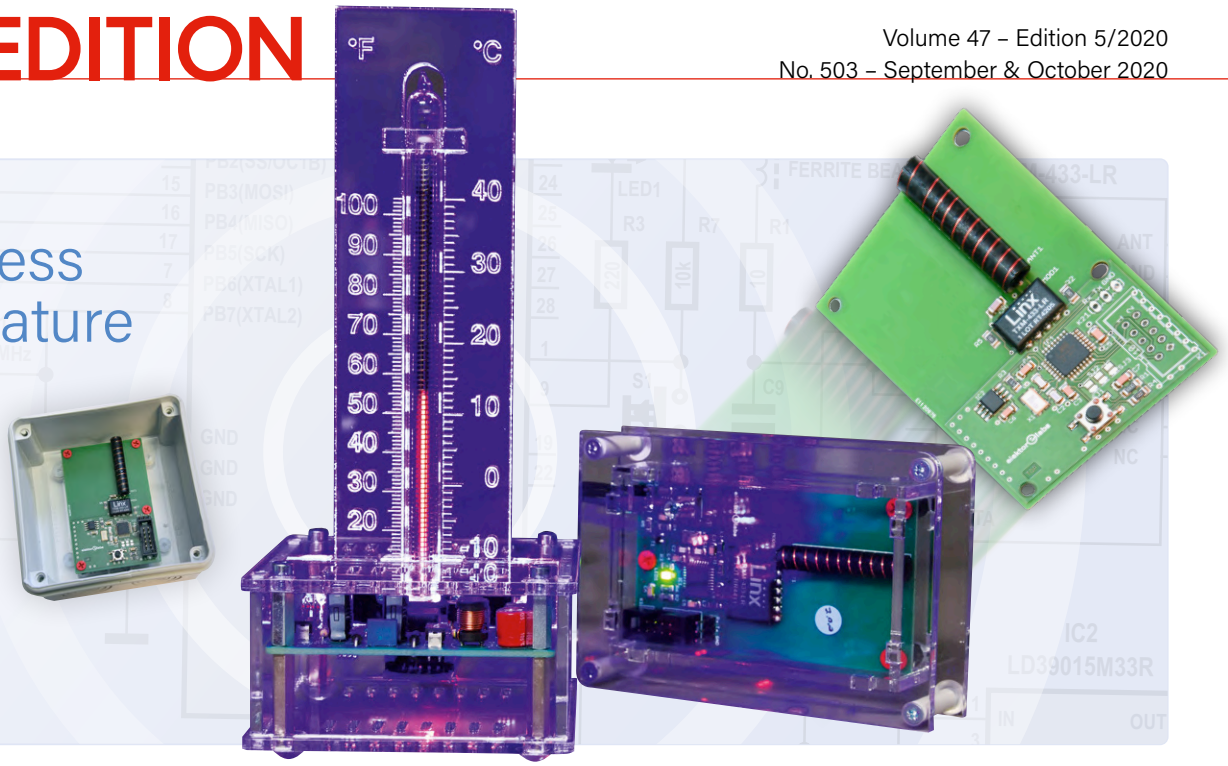
International Editor-in-Chief:	Jens Nickel
Membership Manager:	Denise Bodrone
International Editorial Staff:	Eric Bogers, Jan Buiting, Stuart Cording, Rolf Gerstendorf, Denis Meyer, Dr Thomas Scherer, Clemens Valens
Laboratory Staff:	Mathias Claussen, Ton Giesberts, Luc Lemmens, Clemens Valens, Jan Visser
Graphic Design & Prepress:	Giel Dols, Harmen Heida
Publisher:	Don Akkermans

THIS EDITION

Volume 47 – Edition 5/2020
No. 503 – September & October 2020

Labs Project

A Wireless Temperature Sensor



16

Regulars

Features

Projects

IKEA Lamp Hack
With NeoPixel-LEDs and WLAN

56

Review

Programmable DC Electronic Load

89

Multitasking with Raspberry Pi

23

Next Edition

Elektor Magazine Edition 6/2020 (November & December)
As usual, the next issue is packed with circuits, projects, foundations and tips and tricks for electronic engineers.

From the contents:

- Precise LCR Meter
- LoRa Tracker
- Current Probe
- Power Analyser
- Nixie Watch
- Home Automation with Home Assistant (2)
- FFT on the Maixduino
- Teensy 4.0 - Full speed ahead!
- Battery Management Basics

And much more!

Elektor Magazine edition 6/2020 covering November & Decemeber is published around the 5th November 2020. Arrival of printed copies with Elektor Gold Members is subject to transport. Contents and article titles subject to change.

190365-E-04

USB to S/PDIF Interface

Digital audio output for computer, laptop, tablet or smartphone

By **Stephan Lück**

Most PCs, laptops, tablets and smartphones don't have a 'real' audio output. Instead, they are only provided with a multifunction USB connector and a headphone output. This is a problem when you want to connect one of these devices to a high-quality amplifier or an AV receiver. To solve this problem once and for all we present a high-quality USB to S/PDIF interface.

PROJECT DECODER

Tags
Digital audio, PC, laptop, tablet, smartphone

Level
entry – intermediate – expert

Time
About 4 hours

Tools
Soldering tools (SMD and through-hole), mechanical tools

Cost
€35 / £30 / $40 approx.

SPECIFICATIONS

- Power supply voltage +5 V
- Current consumption 36 - 43 mA
- Microcontroller PIC32MX27F256B-I
- Supports 3 sampling frequencies: 44.1, 48 and 96 kHz
- Supports 16-, 20- and 24-bit audio
- Optical and electrical S/PDIF outputs
- IR remote control (RC5 in the present firmware)
- Works with Windows 7/10, Linux, Android and Raspbian

S/PDIF (or S/P-DIF) means "Sony/Philips Digital Interface Format" and is an interface for transferring audio data in digital form over short distances between modules and systems, such as in home theatre or hifi equipment. It is a uni-directional interface for serial data without a separate clock signal (the clock is recovered from the signal itself). The underlying idea is to keep the audio data in the digital domain for as long as possible and convert it into an analogue signal at the latest conceivable moment.

S/PDIF [1] is based on the professional AES3 standard. At the protocol-level both standards are compatible, but their electrical characteristics are different. A typical professional AES3 interface uses a 3-pin XLR connector and a screened, symmetrical, twisted-pair cable with an impedance of 110 Ω. The signal level ranges from 3 to 10 V_{pp} (AES/EBU 2 to 7 V_{pp}). Less commonly found are implementations using BNC connectors with 75 Ω coax cables. The symmetrical AES3 with XLR connectors (shielded twisted pair) can be used over distances of up to 1000 m; for the coax version this drops to 100 m.

Two types of connection are defined for S/PDIF. The simplest variant (that is generally used in the 'better' audio components) has a 75 Ω coax cable with RCA plugs that are usually coloured orange. The signal level is about 0.5 V_{pp}.

The second variant has the name Toslink (from Toshiba Link). This is a standardised optical fibre with a red LED (659 nm) as the transmitter. Normally this optical fibre is a simple POF cable (Plastic Optical Fibre). This can be used for distances of up to 10 m. With a high-quality glass optical fibre, a distance of up to a maximum of 30 m is possible without repeaters. The optical signals have the same signal structure as the electrical signals – and that is rather obvious since the electrical signal only turns the LED on and off.

The project

The circuit that we describe here is a USB to S/PDIF converter. The converter is easily connected to a PC or similar devices using the USB connector. The S/PDIF output allows it to be connected to AV receivers, high-end amplifiers, or stand-alone audio DACs. The S/PDIF output is implemented twice (both electrical and optical) and remote control is possible thanks to an IR-receiver. **Figure 1** shows the completed circuit.

The S/PDIF bit stream is generated entirely in software by the same microcontroller that is also responsible for the USB connectivity. Here we have, in fact, a single-chip solution that can be realised with relatively little

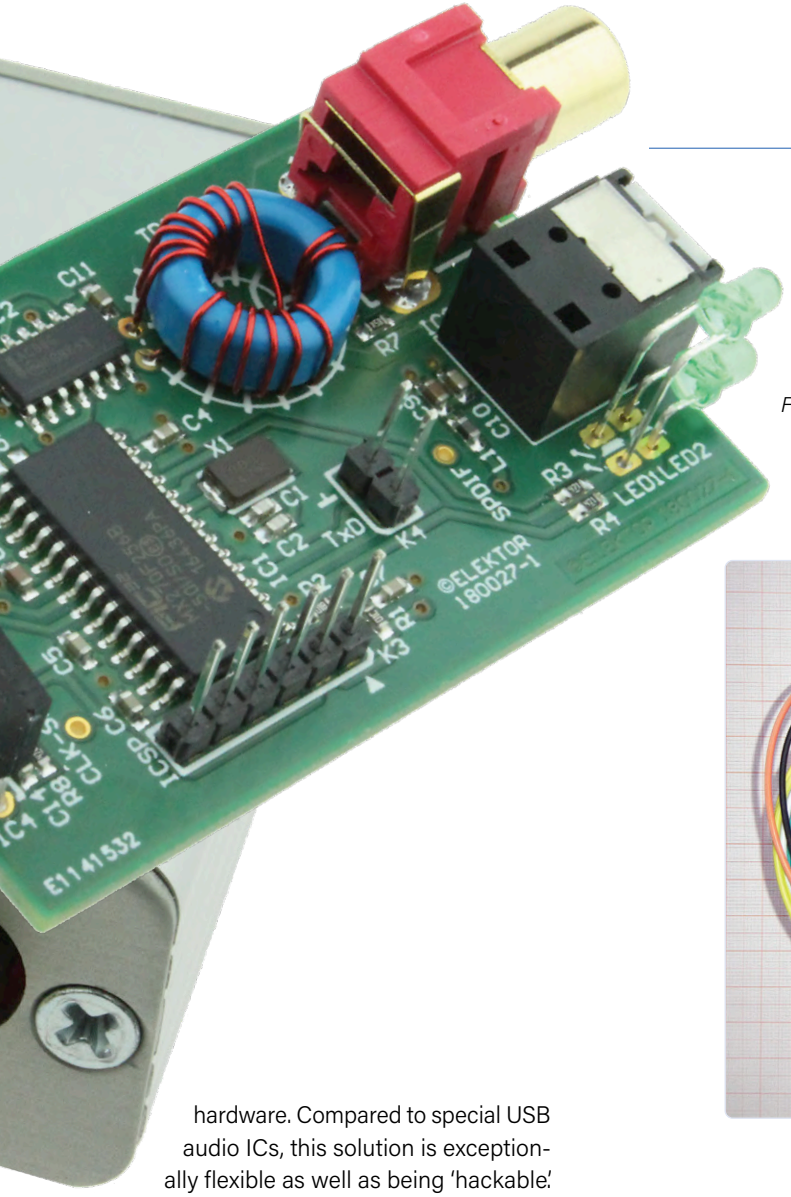


Figure 1: The USB to S/PDIF interface is nice and compact.

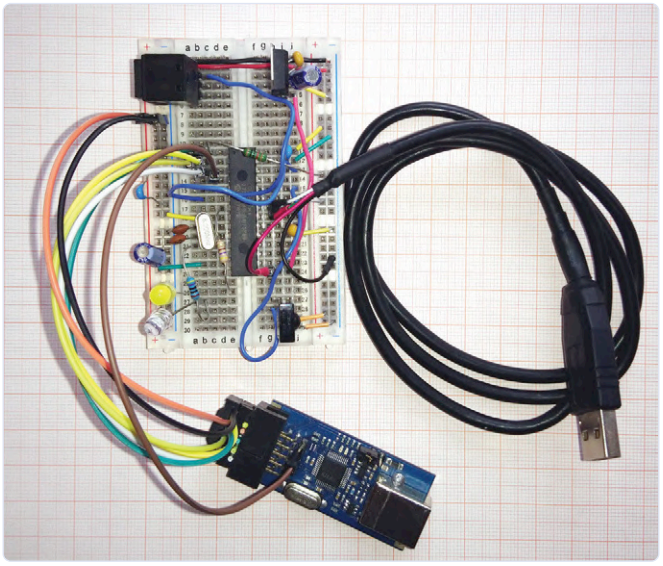


Figure 2: The prototype on a breadboard.

hardware. Compared to special USB audio ICs, this solution is exceptionally flexible as well as being 'hackable.' Furthermore, the construction is not extremely critical – it is even possible to build and operate the circuit on a breadboard (see **Figure 2**).

For this project we selected a PIC32MX270 microcontroller. This has all the peripherals for USB audio applications on board and has sufficient RAM to store the coded S/PDIF frames. In addition, this controller is available in variants with relatively few pins, which greatly simplifies the circuit board design.

A small, double-sided printed circuit board has been designed for this project. Besides the microcontroller it also provides space for the power supply, the optical and electrical S/PDIF outputs, and the receiver for the IR remote control. There are also two LEDs that indicate the audio sampling frequency and whether the output is active. The circuit board has been designed so that it all fits neatly into a small Hammond enclosure.

The audio section

The audio section of the circuit consists of a USB interface conforming to the USB audio class [2][3], the software implementation of the S/PDIF encoder, and the S/PDIF output. This is implemented using the SPI output of the microcontroller together with a DMA channel that continuously copies the S/PDIF frames from the circular buffer to the SPI. **Figure 3** shows the block diagram of the software and hardware components.

Because the standard USB audio specification has been implemented, there is no need to install any special drivers on the host for this USB to S/PDIF interface.

The *device class* specification is quite complex. That is why the USB-IF (*USB Implementers Forum*) has published the *USB Audio Device Class Specification for Basic Audio Devices* [4] that contains a small subset of the original *Audio Device Class* specification. The USB audio interface that was implemented in our project has a strong resemblance to the headphone application that is described in [4]. We have, however, added a couple of additional sampling frequencies, omitted the volume control, and the USB descriptor has been changed so that it describes an S/PDIF output instead of a loudspeaker. Our USB audio interface has the following characteristics:

- Support for three sampling frequencies (44.1 kHz, 48 kHz and 96 kHz);
- One audio format: two channels with 3 bytes (24 bits) per channel (S24_3LE);
- An option for muting the audio signal.

These features are specified in the USB configuration descriptor (found in the file `usb_descriptors.c` in the software archive). Further information about this can be found in [3], [5] and [6].

Because we are not aiming for formal USB compliance, we have selected vendor and product IDs that can be used without the risk of creating a conflict with official USB products.

S/PDIF encoder and serial output

The USB interface uses the isochronous USB endpoint 1 to transfer audio samples to the S/PDIF interface. That means that each USB

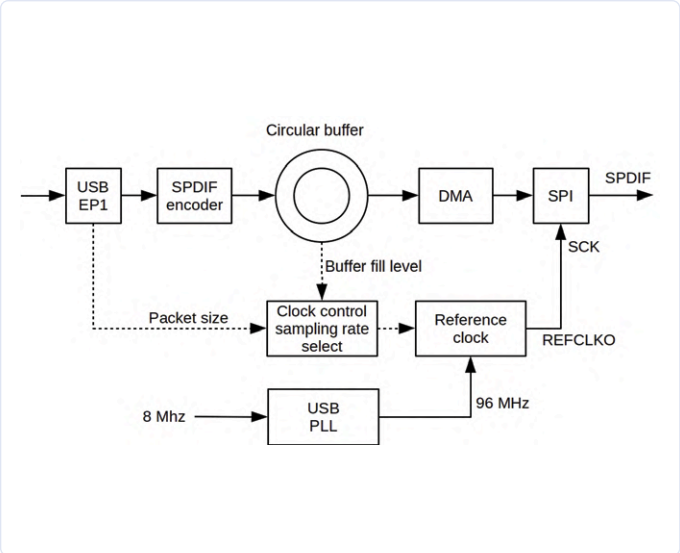


Figure 3: The block diagram for the interface.

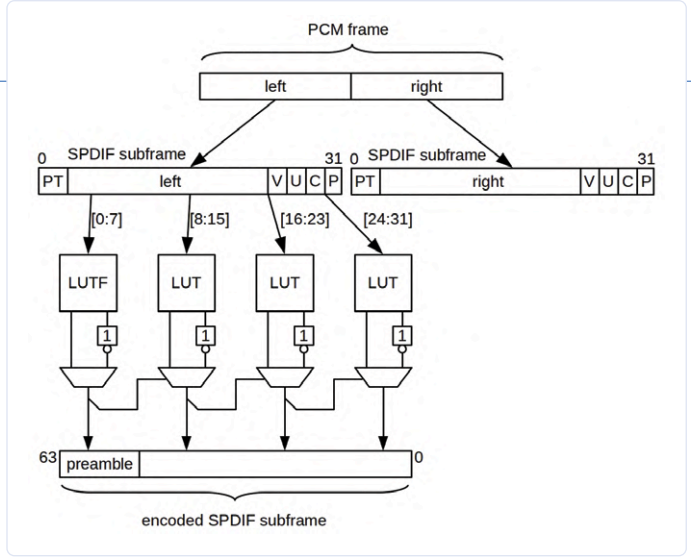


Figure 4: The operating principle of the software S/PDIF encoder.

frame of 1 ms transfers one USB data packet. Each time the USB hardware of the microcontroller has received a new isochronous packet, the interrupt routine is called which converts the individual PCM frames in the USB data packet into corresponding S/PDIF frames. For more information about S/PDIF we refer you to [7] and [8].

The block diagram in **Figure 4** shows how the software of the S/PDIF converter works. At the top of the figure we see the 48-bit PCM frame. Each frame contains two samples – one for the left channel and one for the right channel. These samples are converted into an intermediate 32-bit representation of the S/PDIF subframes by adding a 4-bit preamble tag that specifies the type of S/PDIF preamble (X, Y or Z) that must be used for the final encoding of the S/PDIF subframe. Additionally, the four bits V, U, C and P are added to the end. The V and U bit correspond to the validity and user data bits respectively. In our implementation these bits are always 0. The C bit contains the channel status, while the P bit implements the parity bit for the individual subframes. These parity bits are calculated for each subframe with the aid of the optimised algorithm that is described in [9].

Two successive S/PDIF subframes form one S/PDIF frame. 192 successive S/PDIF frames together form an S/PDIF block. The channel status is obtained by stringing all the C bits of one block one after the other. As a result, the channel status block has a length of 192 bits (24 bytes). Here we use the channel status bits to indicate the current sampling frequency to the receiver that is connected to the S/PDIF output.

S/PDIF uses biphase mark encoding (also known as differential Manchester encoding) to encode the audio data and preamble bits. Because two code bits correspond to one data bit, the encoded S/PDIF subframe comprises 64 bits. Using the simplest implementation of biphase mark encoding, each bit of an S/PDIF subframe would have to be encoded individually, which would of course 'eat' valuable processor time. This is why we use two different look-up tables (LUTs) to encode the data one byte at a time. The LUTF lookup table is used to encode the first byte of an S/PDIF subframe while the LUT lookup table is used for encoding the remaining three bytes of the subframe. The special LUTF table contains the required preamble bit patterns.

The lookup table LUT is drawn three times in **Figure 4** because it is used three times to encode one S/PDIF subframe. In reality, there is only one instance of this table stored in memory. The lookup tables are generated by an initialising function that runs once after the microcontroller comes out of reset.

In the biphase mark code there has to be a transition (edge) between any pair of code bits that represent one data bit. To ensure this, the 16-bit code words from the lookup tables are inverted or not depending on the last bit of the preceding code word.

The lookup tables perform a second function: the reversal of the bit order. In the S/PDIF standard, the subframes are transmitted starting with the LSB, but the SPI peripheral transmits them in the opposite order, MSB first. The LUTs therefore supply the bits in the reversed order so that they will be correctly transmitted by the SPI hardware.

The 64-bit S/PDIF code words that are obtained this way are placed in a circular buffer (in the SRAM memory of the controller) and sent out through the SPI port using DMA.

The S/PDIF bit clock

The programmable fractional frequency divider that is integrated into the microcontroller derives the S/PDIF bit clock from an internal 96-MHz clock signal. The required S/PDIF clock frequency is:

$$f_{SPDIF} = 128 * f_S$$

where f_S is the audio sampling frequency.

The USB audio specification does not allow for explicitly indicating the sampling frequency f_S that is used by the host to the USB device. This means that the microcontroller software must determine it based upon the number of PCM frames in the isochronous USB packets. Note that the USB descriptor only contains three allowable sampling frequencies of 44.1 kHz, 48 kHz and 96 kHz. The host is not allowed to use any other sampling frequencies. One advantage of this is that the microcontroller can simply 'guess' the sampling frequency chosen by the host based on the number of PCM frames in a USB packet.

Clock perils

In compliance with the USB Basic Audio Devices specification [4], the 'synchronous' synchronisation type was selected for the isochronous audio endpoint. Or, expressed differently, the sampling speed of the audio stream is dependent upon the USB clock domain and the frequency with which the SOF tokens are generated by the host. On the output side, the S/PDIF clock is derived from the local oscillator. To prevent over- or

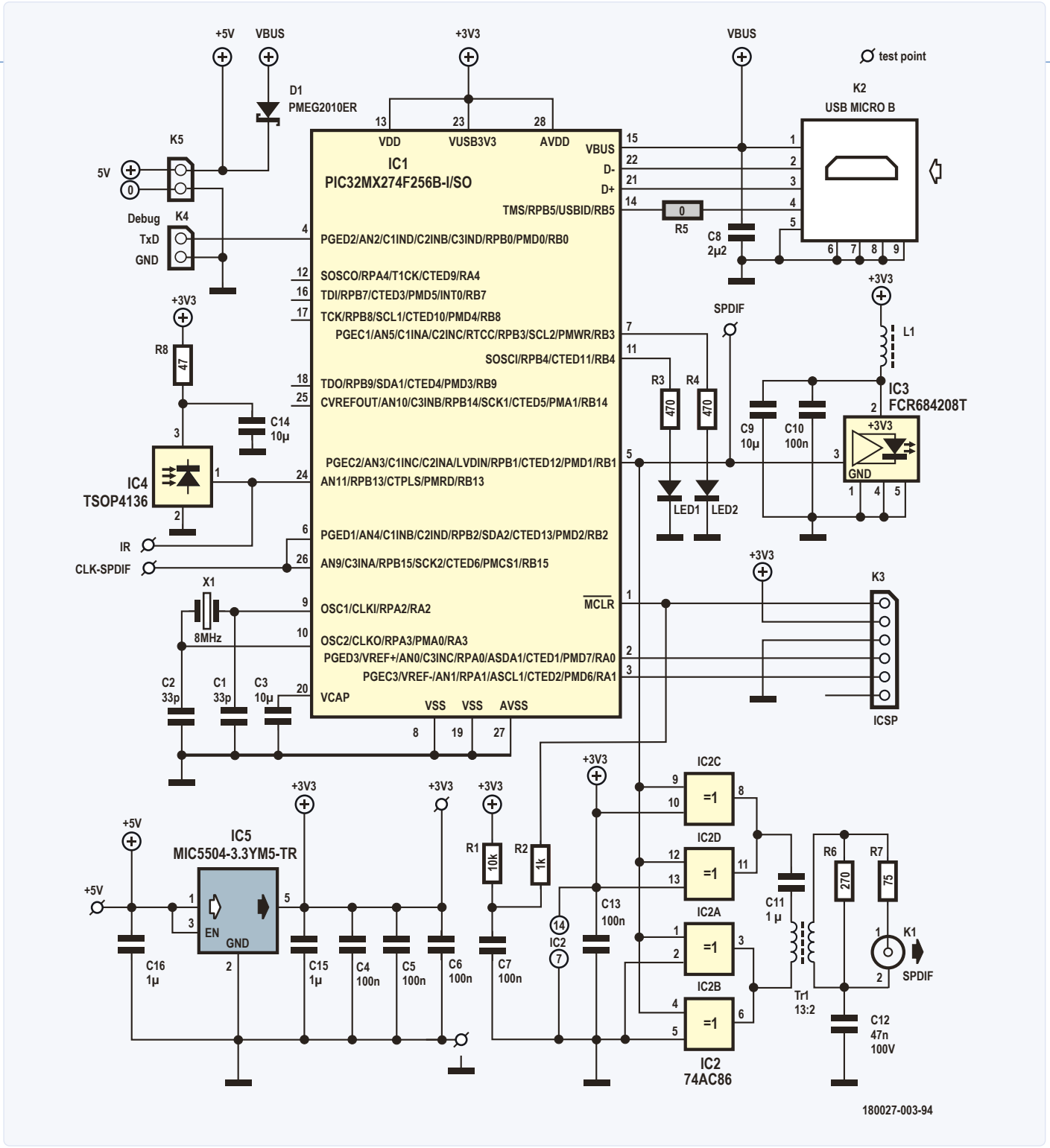


Figure 5: The full schematic for the USB to S/PDIF interface.

under-flow of the circular buffer the software frequently changes the fractional part of the frequency divider based on the average fill-level of the circular buffer.

Although Microchip's Application Note AN1422 [10] explicitly states that the reference clock can be adjusted during operation, we had problems when changing the divider ratio for cases where this value was small (more specifically, when a sampling frequency of 96 kHz was selected). Every time the divider ratio was changed, a short interruption in the clock signal occurred that resulted in audible artefacts.

To solve this problem we made an external connection between the output of the reference clock oscillator REFCLKO (pin 6) and the SPI clock input SCK (pin 26). This allowed the baud rate generator of the SPI to be circumvented.

IR remote control

The IR remote control receiver is entirely independent of the audio section of the circuit. In other words, everything that is received by the receiver is directly passed on to the host without directly influencing the audio section.

For decoding the IR signal of the remote control, use is made of the open source IRMP library [11]. This allows many different IR signals to be decoded. The microcon-

COMPONENT LIST

Resistors (all SMD 0603, 1%, 0.1 W)
R1 = 10 kΩ
R2 = 1 kΩ
R3, R4 = 470 Ω
R5 = 0 Ω not fitted
R6 = 270 Ω
R7 = 75 Ω
R8 = 47 Ω

Capacitors (all SMD 0603)
C1, C2 = 33 pF, 50V, 5%, C0G/NP0
C3, C9, C14 = 10 µF, 16V, 20%, X5R
C4 - C7, C10, C13 = 100 nF, 50V, 10%, X7R
C8 = 2.2 µF, 10V, 10%, X7R
C11, C15, C16 = 1 µF, 50V, 10%, X5R
C12 = 47 nF, 100V, 10%, X7R

Inductors
L1 = 600 Ω @ 100 MHz, 0.15 Ω, 1.3 A, SMD 0603 (Murata, BLM18KG601SN1D)
TR1 = toroidal core, 12.5x5mm, material T38, Epcos B64290L0044X038

Semiconductors
LED1, LED2 = LED, green, 3 mm, through-hole
D1 = PMEG2010ER, SMD SOD-123
IC1 = PIC32MX274F256B-I/SO, SMD SO-28
IC2 = 74AC86, SMD SO-14
IC3 = FCR684208T, Toslink (Cliff Electronic Components)
IC4 = TSOP4136
IC5 = MIC5504-3.3YM5-TR, SMD SOT-23-5

Miscellaneous
K1 = RCA jack for circuit board mounting, through-hole (Pro Signal, PSG01545)
K2 = micro-USB type B, female, SMD (Molex, 47346-0001)
K3 = 1 x 6 pin-header, vertical, pitch 2.54 mm, through-hole
K4, K5 = 1 x 2 pin-header, vertical, pitch 2.54 mm, through-hole
X1 = 8 MHz crystal, 5 x 3.2mm, SMD (Abracon, ABM3-8.000MHZ-D2Y-T)
TR1 = 0.3 m varnished copper wire (CuL), Ø 0.5 mm
Enclosure Hammond 1455D601, 60 x 42.5 x 23 mm

PCB 180027-1 v2.01 (Elektor Store)

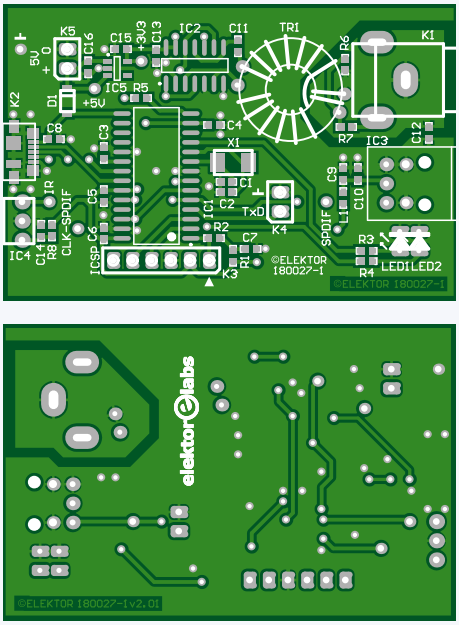


Figure 6: A compact circuit board has been designed for the application.

troller software includes a USB HID implementation [12] that makes it possible to send codes as specified in the HID Usage Tables document [13].
The connection between the IRMP and HID implementation is obtained using a keymap table that couples the codes detected by the IRMP to the usage IDs as specified in [13]. We decided to add the codes conforming to the RC5 standard. Although this standard is being used less often, it is nevertheless readily available on many universal remote controls. The overall thinking was that it would work well in many situations without creating interference with other devices such as TVs and the like. A few of the RC5 codes are listed in [14]. The keymap table also contains codes for a Denon remote control.
The keymap is in the source file `irhid.c` with the name `irhid_keymap`. This can easily be changed to meet your requirements by adding or removing lines and subsequently compiling the software again. To add a button you need to find the IRMP codes (address and command) for the relevant protocol. This can be done by connecting a terminal emulator (115200 bit/s) to the S/PDIF output and pushing the relevant button. You should then be able to see the codes in the terminal emulator. If that doesn't work then you may have to activate the appropriate remote control code by editing the file `irmpconfig.h`. You also need the corresponding USB HID usage ID that is to be coupled to the button. These IDs can be found in [13].

Assembly

After all the explanations above, we can be very brief about the detailed schematic in **Figure 5**. IC1 is the heart of the circuit: the microcontroller. This can be programmed in-circuit via ICSP connector K3. If you don't feel like doing that, then you can order a pre-programmed controller from the Elektor Shop. The power source for the circuit is an external +5 V power supply that is connected to K5.
The USB signals enter via K2 and connect directly to the microcontroller. Zero-ohm resistor R5 must not be fitted because the current software version does not use USBID.
IC4 is a standard IR receiver and IC3 takes care of the optical (Toslink) output. The electrical S/PDIF output signal is routed to K1 via quad buffer IC2. And so we arrive at the 'most curious' component of this circuit: transformer TR1. This is implemented less to provide galvanic isolation and more for preventing earth loops. To minimise RF noise the output has to be decoupled, hence the presence of C12. The XOR gates of IC2 function as a full bridge with the added advantage of increasing the primary voltage and providing better coupling to the secondary winding. The turns ratio of the transformer is 13:2 resulting in a voltage that is practically exactly 0.5 V_{pp} into a load of 75 Ω.
A (double-sided) printed circuit board, **Figure 6**, has been designed for the circuit. Although most of the components are SMD types, assembly shouldn't be much of a problem for the more experienced soldering 'l'artiste.'
The circuit board fits into an aluminium Hammond enclosure, type 1455D601 (60 x 42.5 x 23 mm). It simply slides into place as can be seen in **Figure 7**. Depending on any manufacturing tolerances, it may be necessary to run a file along the edges of the circuit board.
When using the Hammond enclosure specified in the parts list, it is best if you do not use either of the black plastic bezels. It was found that micro-USB connectors end up too deep inside the enclosure for the corresponding plug to make proper contact. The side panels do, of course, require the drilling and cutting of suitable openings for the connectors and LEDs. Talking about the LEDs: their wires can be bent such that they 'look' to the outside, one above the other. **Figure 7** clearly shows

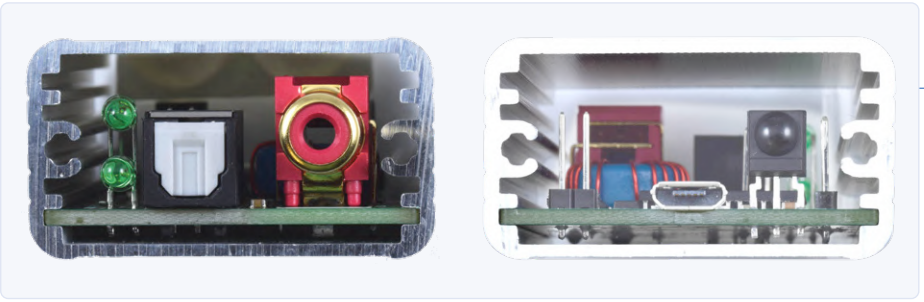


Figure 7: The assembled circuit board slides into the aluminium enclosure.

the intended implementation.
A note with regard to the transformer: unfortunately there is no way to avoid having to wind this component yourself on the toroidal core that is specified in the parts list. As can be seen in **Figure 8**, the primary winding (13 turns) is effectively wound in two halves on the toroid. This has the advantage that the connections for the primary and secondary windings are opposite each other. The secondary has only two turns and even the greatest despisers of coil winding will have no difficulty with this. For the winding wire you need 0.5 mm varnished copper wire; around thirty centimetres should be enough.

A few practical remarks

We have, of course, thoroughly evaluated the USB to S/PDIF interface in our lab and have tried it with various operating systems. It proved to work well with Windows 7 and Windows 10, Linux (Lubuntu and Kubuntu), Android, and even Raspbian on a Raspberry Pi 3 model B+ when using *2019-09-26-raspbian-buster-full.img*.

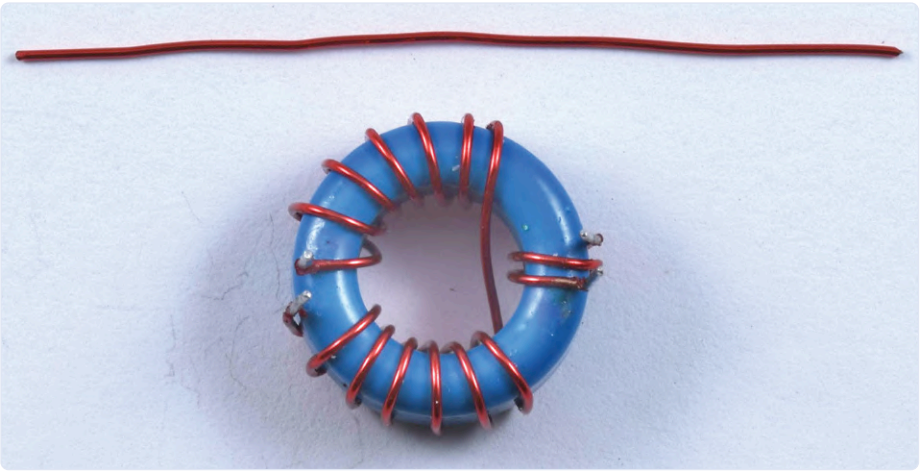


Figure 8: This is how to wind the toroidal transformer. Note that the primary winding is effectively split into two halves.

Windows

Windows automatically recognised the interface. The sampling frequency can be selected in the Control Panel under the option Sound. Select 'SPDIF Interface Properties' from the tab page 'Advanced' and you can select the format, as shown in **Figure 9**.

Lubuntu

With Linux (we used Lubuntu) the sampling frequencies depend on the file. When using the simple GNOME MPlayer a 44.1 kHz file appears as 44.1 kHz on the S/PDIF output. However, even though the output had already been configured, it was necessary to select 'USB_SPDIF Stereo (IEC958) (PulseAudio)' as the Audio Output every time a file was played. This is very annoying. This can be avoided by turning off all other audio devices (here: Sound & Video – PulseAudio Volume Control – Configuration). 48 kHz and 96 kHz files were both played as 48 kHz. Another alternative is to use ALSA directly, without PulseAudio, in a terminal.
If you don't hear anything, first start alsamixer in a terminal (Ctrl+Alt+T and enter `alsamixer`). Press F6 to select the sound card 'USB_SPDIF' (only one small control element appears that shows 00 and PCM, because the interface does not have a volume control). Then close the mixer and enter:

```
aplay -D plughw:CARD=USBSPDIF //.../
```

where `//.../` stands in for the actual path to the audio file.

This can only be used to play non-compressed wav files. To play mp3 files from the command line you can use, for example, mpg123, but there are plenty of other possibilities.

Although USB Audio DACs have been around for many years, the standard method that Linux uses to handle audio files is still in much need of improvement. Setting the output sampling frequency is also possible by editing the configuration files for PulseAudio (`/etc/pulse/daemon.conf`). Information about this can be found online. However, first make a backup of the original file – it is far too easy to make a mistake!

Android

Connect the interface to the device using an OTG cable. Installing the app named 'USB Audio Player Pro' is probably the easiest way to use our interface. This circumvents the audio limitations of Android. All three sampling frequencies of the interface are supported. Once the app has started the interface is recognised immediately.

Raspbian

Our final test was to try the interface with Raspbian. Just like Lubuntu, Raspbian is a free operating system that is also based on Debian. It is therefore unsurprising that the test experience was similar to that using

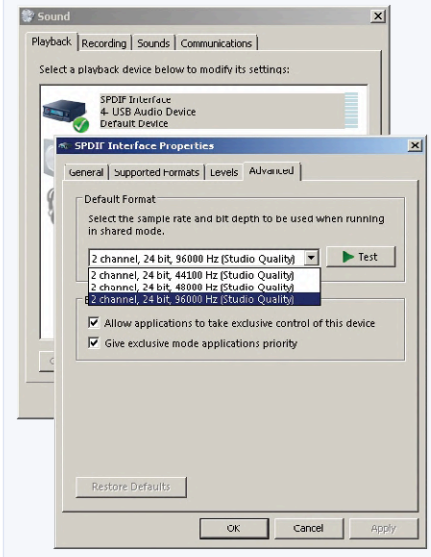


Figure 9: Configuration settings for Windows 7.

SHOPPING LIST

- **USB to S/PDIF interface, bare board:**
www.elektor.com/180027-1
- **USB to S/PDIF interface, pre-programmed microcontroller:**
www.elektor.com/180027-41

Lubuntu. Playing of uncompressed wav files is undertaken as follows:

```
aplay -D plughw:CARD=USBSPDIF //.../
```

Pressing Ctrl+c stops playback.
When playing mp3 files from the command line, mpg123 worked with our fresh installation of Rasbian only after PulseAudio and its volume control were also installed. If it is not already installed, mpg123 can be installed using:

```
sudo apt-get install mpg123
```

PulseAudio can be installed as follows:

```
sudo apt-get install pulseaudio
```

Installation of the accompanying volume control is performed using:

```
sudo apt-get install pavucontrol
paprefs
```

Make sure to restart the system after installing everything.
Just as with Lubuntu, the onboard audio in Sound & Video – PulseAudio Volume Control – Configuration has to be turned off .
The pre-installed VLC Media Player only produced sound after PulseAudio was also installed. A file sampled at 32 kHz will be played at 96 kHz.
We have, of course, also measured the audio performance. For a description of these measurements and their results we refer you to the Elektor Labs page for this project [15].
At the moment of writing, the current software can be downloaded from the project page for this article [16] or from GitHub [17].
That only leaves us to wish you much building and listening pleasure! As always, your observations and criticisms are gratefully received.

180027-03

WEBLINKS

[1] **Details for S/PDIF and Toslink (German)**: https://kompendium.infotip.de/spdif_toslink.html
[2] **Universal Serial Bus Specification Revision 2.0**: https://www.usb.org/document-library/usb-20-specification
[3] **Universal Serial Bus Device Class Definition for Audio Devices, Release 1.0**: www.usb.org/document-library/audio-device-document-10
[4] **Universal Serial Bus Audio Device Class Specification for Basic Audio Devices, Release 1.0**: www.usb.org/document-library/audio-device-class-spec-basic-audio-devices-v10-and-adopters-agreement
[5] **Universal Serial Bus Device Class Definition for Audio Data Formats, Release 1.0**: www.usb.org/document-library/audio-data-formats-10
[6] **Universal Serial Bus Device Class Definition for Terminal Types, Release 1.0**: www.usb.org/document-library/audio-terminal-types-10
[7] **AES3-2003:** AES standard for digital audio – Digital input-output interfacing – Serial transmission format for two-channel linearly represented digital audio data
[8] **Crystal Application Note AN22**: "Overview Of Digital Audio Interface Data Structures": https://statics.cirrus.com/pubs/appNote/an22.pdf
[9] **Sean Eron Anderson: "Bit Twiddling Hacks", section "Compute parity in parallel"**: http://graphics.stanford.edu/~seander/bithacks.html#ParityParallel
[10] **Microchip AN1422**: "High-Quality Audio Applications Using the PIC32": http://ww1.microchip.com/downloads/en/AppNotes/01422A.pdf
[11] **Infrared Multi-Protocol decoder (IRMP) (German)**: https://www.mikrocontroller.net/articles/IRMP
[12] **USB Device Class Definition for Human Interface Devices (HID), Version 1.11**: www.usb.org/document-library/device-class-definition-hid-111
[13] **USB HID Usage Tables, Version 1.12**: www.usb.org/document-library/hid-usage-tables-112
[14] **IR Remote Control Codes (1) - Elektor March 2001**: www.elektormagazine.com/magazine/elektor-200103/16978
[15] **Project page at Elektor Labs**: www.elektormagazine.com/labs/usb-spdif-interface-180027
[16] **Project page for this article**: www.elektormagazine.com/180027-03
[17] **Software download on GitHub**: https://github.com/kiffie/usb-spdif

Practical
ESP32 Multitasking (4)
Binary semaphores

By **Warren Gay** (Canada)

When multitasking in FreeRTOS, there is often a need to synchronize between tasks. One such synchronization facility is the binary semaphore. This article explores the binary semaphore and illustrates it with a simple demonstration.

What is a semaphore?

A semaphore is a function that permits the caller to block its continued execution until permission is "given" to proceed. In other words, the caller tries to "take" the semaphore but, if the semaphore is already taken, the calling task waits (blocks). This is much like boys wishing to select a popular girl at a dance. If the girl is currently dancing (taken), then the other interested boys must wait until she is ready to "give" herself again. This leads us to define one of the properties of a semaphore: a *binary* semaphore has two states:

- taken
- given

How does the semaphore protect?

A semaphore itself cannot control access to any resource. It is only by *agreement* that resource protection may be implemented via a semaphore. If the boys at the dance didn't respect the concept of "taken" and "given", then multiple boys could grab the girl by the arms in a struggle. The semaphore operates by protocol: the accessor agrees to use any resource on offer only if it succeeds in taking the semaphore. It will also not use the resource again until it has taken the next available semaphore.

Timeouts when taking

A boy waiting at the dance may have limited patience. He may decide that, after ten minutes of waiting, he will try for a different girl to dance with. Binary semaphores also allow the caller to specify such a timeout period in system ticks. If an attempt to take a semaphore takes longer than the specified number of ticks, the call will return with a fail code. Alternatively, FreeRTOS can be told to wait indefinitely until the semaphore has been given (much like a boy that is smitten). Finally, there is the option to specify no timeout at all with the attempt failing immediately if the semaphore cannot be taken.

To summarize, semaphores can operate with time in three ways:

- Block forever, until the semaphore can be "taken" by the caller.
- Block for a defined period of time, failing if the operation times out.
- Fail immediately if the semaphore is currently "taken".

Ownership and giving

When a semaphore is taken, the task is said to "own" it. The boy who is currently dancing with the girl then effectively "owns" her (but perhaps not her heart). When he has finished dancing with her, he can "give" her to someone else immediately or simply give her hands back, freeing her: there is no waiting involved. Likewise, when "giving" a semaphore, there is no need for a timeout argument. The giving of an owned semaphore happens immediately.

The act of "giving" a semaphore does not *necessarily* imply that it will be immediately "taken" by another task. Once the boy has finished dancing with the girl she becomes available, but that doesn't mean that there are necessarily other interested takers. The other boys may have given up and found different dance partners.

Initial state

Let's strain this analogy a little further – the girl is created in the "taken" state by her parents. Only when her parents allow her to attend the dance is she "given" and becomes available. In the same way, the FreeRTOS binary semaphore is initially created in the "taken" state. This differs from mutexes in Linux or Windows that you may be comparing it to, as they are created in a "given" state. More will be said about the FreeRTOS mutex in a later installment.

xSemaphoreBinaryCreate():

To create a binary semaphore in FreeRTOS is simple:

```
SemaphoreHandle_t h;

h = xSemaphoreCreateBinary();
assert(h)
```

There are no arguments to supply, and a handle to the binary semaphore that was created is returned. It is possible that the handle will be returned as `nullptr` (`NULL`) if you've exhausted the available memory. It is recommended to check the returned value (the `assert()`

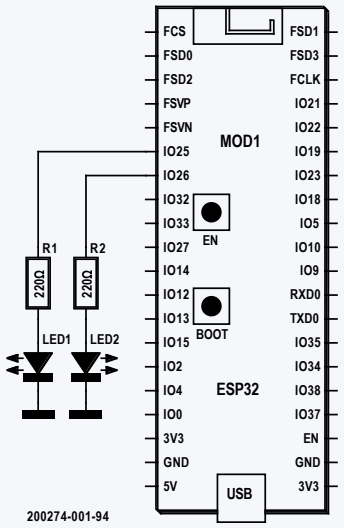


Figure 1. Schematic for the semaphores.ino demonstration.

macro from assert.h was used here) to ensure the semaphore was created.

xSemaphoreGive()

To give a semaphore is also straight forward:

```
SemaphoreHandle_t hMySemaphore;
BaseType_t rc; // return code

...
rc = xSemaphoreGive(hMySemaphore);
assert(rc == pdPASS);
```

The "give" operation can fail, returning `pdFAIL`, only for the following reasons:

- The handle (`hMySemaphore`) is invalid.
- The semaphore has already been "given".

xSemaphoreTake()

Taking a semaphore is potentially a blocking operation for the calling task:

LISTING 1: THE SEMAPHORES.INO DEMONSTRATION PROGRAM [1].

```
0001: // semaphores.ino
0002: // Practical ESP32 Multitasking
0003: // Binary Semaphores
0004:
0005: #define LED1_GPIO   25
0006: #define LED2_GPIO   26
0007:
0008: static SemaphoreHandle_t hsem;
0009:
0010: void led_task(void *argp) {
0011:   int led = (int)argp;
0012:   BaseType_t rc;
0013:
0014:   pinMode(led,OUTPUT);
0015:   digitalWrite(led,0);
0016:
0017:   for (;;) {
0018:     // First gain control of hsem
0019:     rc = xSemaphoreTake(hsem,portMAX_DELAY);
0020:     assert(rc == pdPASS);
0021:
0022:     for ( int x=0; x<6; ++x ) {
0023:       digitalWrite(led,digitalRead(led)^1);
0024:       delay(500);
0025:     }
0026:
0027:     rc = xSemaphoreGive(hsem);
0028:     assert(rc == pdPASS);
0029:   }
0030: }
0031:
0032: void setup() {
0033:   int app_cpu = xPortGetCoreID();
0034:   BaseType_t rc;  // Return code
0035:
0036:   hsem = xSemaphoreCreateBinary();
0037:   assert(hsem);
0038:
0039:   rc = xTaskCreatePinnedToCore(
0040:     led_task,   // Function
0041:     "led1task", // Task name
0042:     3000,       // Stack size
0043:     (void*)LED1_GPIO, // arg
0044:     1,          // Priority
0045:     nullptr,    // No handle returned
0046:     app_cpu);   // CPU
0047:   assert(rc == pdPASS);
0048:
0049:   // Allow led1task to start first
0050:   rc = xSemaphoreGive(hsem);
0051:   assert(rc == pdPASS);
0052:
0053:   rc = xTaskCreatePinnedToCore(
0054:     led_task,   // Function
0055:     "led2task", // Task name
0056:     3000,       // Stack size
0057:     (void*)LED2_GPIO, // argument
0058:     1,          // Priority
0059:     nullptr,    // No handle returned
0060:     app_cpu);   // CPU
0061:   assert(rc == pdPASS);
0062: }
0063:
0064: // Not used
0065: void loop() {
0066:   vTaskDelete(nullptr);
0067: }
```

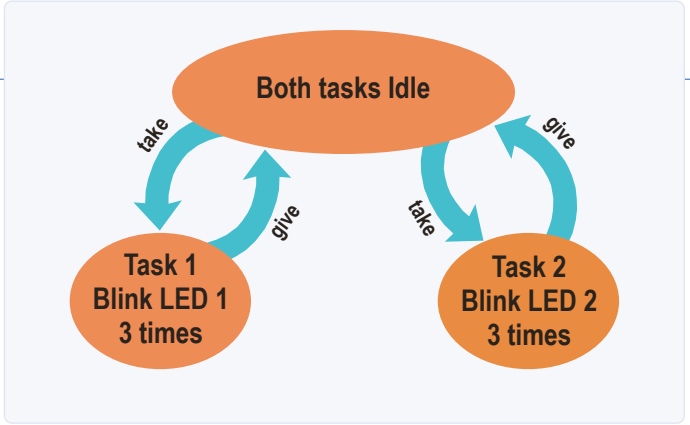


Figure 2. The states of the executing tasks in the semaphores.ino program.

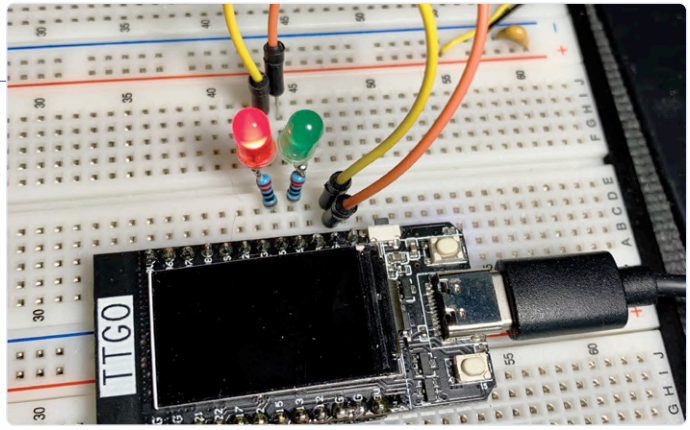


Figure 3. The ESP32 TTGO driving two LEDs using the program semaphores.ino.

```
SemaphoreHandle_t hMySemaphore;
TickType_t wait = 30; // ticks
BaseType_t rc; // return code

...
rc = xSemaphoreTake(hMySemaphore, wait);
// Returns pdPASS when taken,
// otherwise returns pdFAIL if it times out
```

The "take" operation can fail if the handle is invalid or the call has timed out (period as defined by the `wait` argument). Otherwise, the call will block the calling task's execution until it has "taken" the semaphore. The `wait` parameter can have one of three different values according to the caller's requirements:

- Macro value `portMAX_DELAY` – wait forever until "taken".
- Some positive non-zero value – wait for so many ticks.
- Zero – fail immediately if the semaphore cannot be "taken" immediately.

Demonstration

As a demonstration, two tasks will each blink their own LED (see **Listing 1**). By sharing a binary semaphore, only one of the tasks will be allowed to blink their LED at any one time. The non-running task must wait until the semaphore has been "given" by the opposing running task, making the semaphore available to be "taken" once more. **Figure 1** illustrates the schematic for the LED hookup for any ESP32 you choose to use.

Figure 2 illustrates the states of the pair of tasks executing in the demo program. When one task takes the semaphore, that owning task is able to execute and blink its LED, while the other is blocked waiting. Only when the owning task gives the semaphore back can the other task take it and execute. **Figure 3** illustrates an example breadboard setup using the TTGO ESP32 dev board.

In the `setup()` routine, line 36 creates the semaphore and assigns the handle to the static global variable `hsem`. Lines 39 to 46 creates the first task to blink LED1 (note the argument in line 43 of the task creation call). The GPIO pin to be used is passed as a `void` pointer. This is then cast back to an `int` GPIO value in the task function `led_task()` (line 11), of the task function. This is abusing the pointer to pass a value, but this works as long as the value fits within the same number of bits as a pointer address.

After the first task is created, we "give" the semaphore in line 50. By doing this before the creation of the second task makes it likely that the first task acquires the semaphore first. Lines 53 to 60 then creates the second task. Both tasks run the same function code `led_task()` but with different argument values specifying the LED GPIO pin to be used. Lines 14 and 15 configure the LED being used within the task. The task then enters an endless loop starting at line 17. The first step performed within this loop is to "take" the semaphore (line 19). Only one task at a time will succeed at this.

The task that succeeds in taking the semaphore will continue through the loop in lines 22 to 25. This loop blinks the task's LED three times. Once complete, the semaphore is "given" in line 27, allowing the other task to "take" the semaphore. In this way, each task takes turns in handing control over to the other task.

Summary

The binary semaphore used in this demonstration was exercised in two different ways. Before the first "give" operation in the `setup()` function, neither task is able to proceed. This caused the semaphore to behave as a barrier because no task was able to proceed. After that first "give" operation, one of the two tasks "takes" that semaphore, blinks their LED, and then "gives" the semaphore back. Operating in this manner, the semaphore seems to behave like a mutex. Both tasks continually attempt to execute their code but, through the mutual exclusion of the semaphore, only one task is ever permitted to blink its LED.

It is important to memorize that the binary semaphores are created in the "taken" state (unlike a mutex). If the semaphore in this demonstration had not initially been given in the `setup()` routine (line 50), both tasks would have been stuck waiting forever. Other synchronization primitives are available within FreeRTOS (including the mutex), but the simple binary semaphore is sometimes all that is needed.

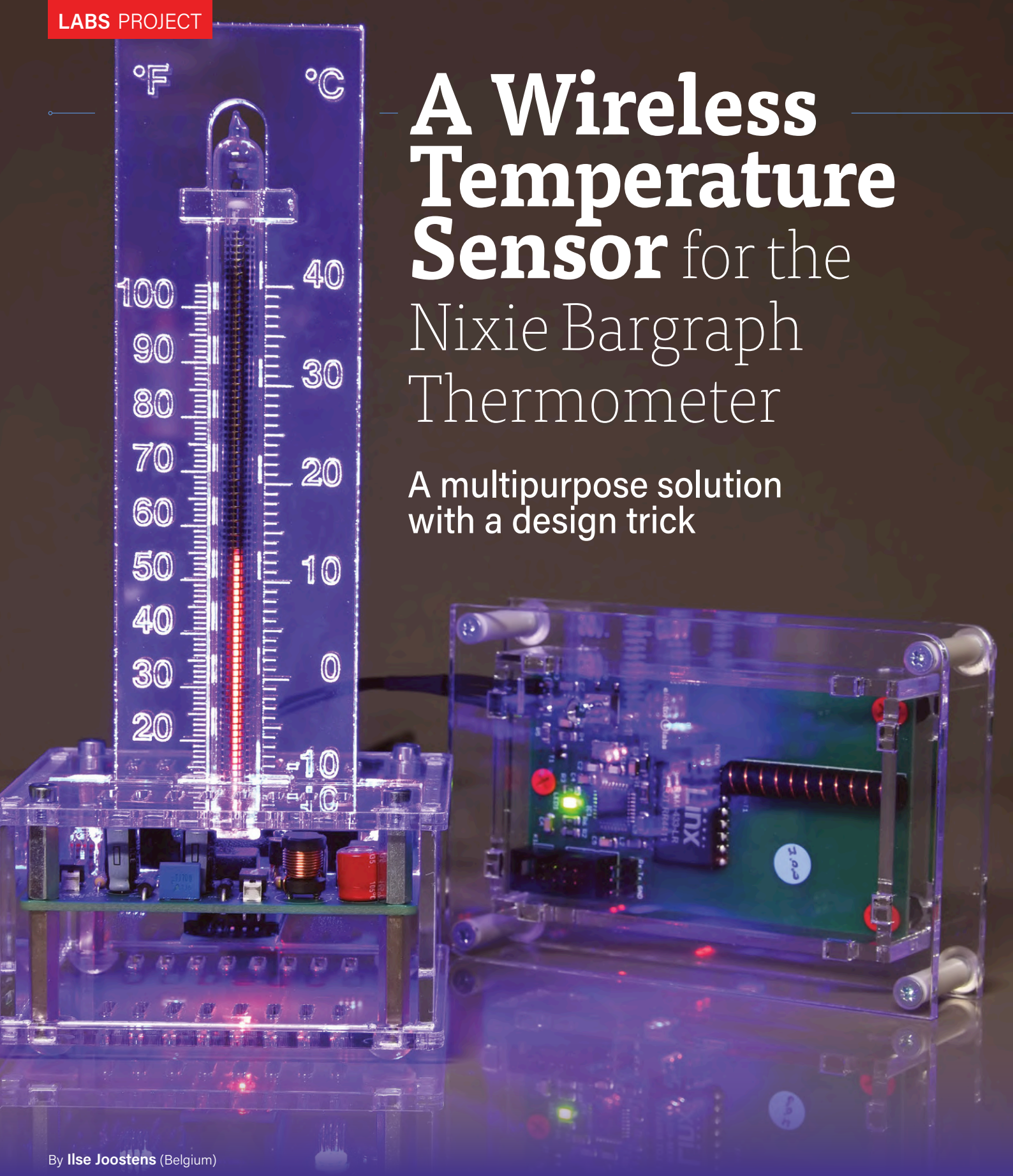
200274-01

WEB LINK

[1] https://github.com/ve3wwg/esp32_freertos/blob/master/semaphores/semaphores.ino

A Wireless Temperature Sensor for the Nixie Bargraph Thermometer

A multipurpose solution with a design trick



By **Ilse Joostens** (Belgium)

Having digested Elektor's Nixie Bargraph Thermometer article [1] one reader based in the American Midwest kindly advised that temperatures in his area easily exceed 85 °F (29.5 °C) for more than six months. Under these circumstances the original 50–85 °F (10–30 °C) scale of our thermometer is inconvenient if not useless. In response and in good engineering spirit we expanded the scale to cover 15–105 °F (–10 to +40 °C) and while at the design desk, we thought we'd turn that thermometer into a wireless device as well. Here goes.

PROJECT DECODER

Tags
433 MHz, Linx, LPR, 1-Wire, Manchester, Nixie, sensorics

Level
entry level – intermediate level – expert level

Time
1.5 hours approx.

Tools
lab tools, solder iron

Price
€75 approx.

FEATURES

- 3.3–5 V supply (receiver)
- 3 V supply (transmitter on CR2450 battery)
- 433.92 MHz licence-exempt radio link (USA/CAN: 315 MHz)
- TX-on interval approx. 2 minutes, adjustable
- Integrated half-wave helical PCB antenna
- DS2438Z temperature sensor
- Usable temperature range: –20 °C to +70 °C (±2 °C accuracy)
- Linx Technologies OOK/ASK RX, TX modules
- RX emulates DS18B20 1-Wire temperature sensor but down to 2.4 V
- Pre-assembled RX and TX boards

Initially we thought it would be more difficult to read the temperature on the extended scale, the readout of the IN-9 bargraph tube being delightfully imprecise. In practice though it is not too bad. Having such a scale on the Nixie Bargraph Thermometer [1] is also useful to display the outdoor temperature if you live in a region with a more moderate climate than our US correspondent, hence the idea to design wireless connectivity into the temperature sensor.

Wireless receiver and transmitter modules

Nowadays many types of wireless receiver and transmitter modules are available off the shelf at prices ranging from a few euros/dollars/pounds up to several tens, for a set consisting of a receiver (RX) and a transmitter (TX). For the sake of simplicity and in order to achieve a reasonable distance between the transmitter and the receiver we opted for 433-MHz modules, i.e. type-approved and licence-free.

To keep the price of the finished product as low as possible, we tested the notorious Chinese FS1000A and MX-RM-5V modules first. We bought three pairs of transmitter and receiver modules on eBay, then fabricated prototype PCBs for a test setup. Ultimately, the modules worked but the range was limited at best. The major problems encountered were:

1. the transmitter requiring 9–12 V to deliver enough power to the antenna;
2. the receiver module being extremely sensitive to noise on the supply rail.

We then looked at other (Chinese) RF modules and found that most of them using simple ASK or OOK modulation (on-off keying / amplitude-shift keying) originated from obscure manufacturers without any guarantee concerning future availability. Another option was to use transceiver modules with an SPI interface but then we would have had to rewrite the firmware which was almost finished at that point.

After another mishap with some 433 MHz FM modules, which worked well but turned out to have the status "not recommended for new designs" **well after order acceptance and delivery** we finally stumbled upon the TXM-433-LR / RXM-433-LR pair from Linx Technologies. These may not be the cheapest modules around but they are easily available in larger quantities and you will save money on batteries as the transmitter in particular is very energy efficient. The Linx RF modules are also available in 315 MHz and 418 MHz versions which permits use in the USA, Canada and other territories where these bands have an ISM (industrial/scientific/medical) allocation.

> Trick: DS2438Z+ = DS18B20 down to 2.4 V!

Transmitter — hardware

Glorious at the heart of the transmitter circuit diagram in **Figure 1** sits an ATMEGA328P-AU microcontroller ticking at 8 MHz using a crystal for its oscillator. And yes, that's the very same microcontroller widely used on Arduino Uno and Nano boards.

At first we thought we'd go for the type DS18B20 1-Wire temperature sensor but it wasn't a good candidate for a 3 V battery powered device as the device's minimum supply voltage is … 3 V according to the datasheet [2]. While there are plenty of other temperature sensors around that work happily at supply voltages under 3 V, we preferred a 1-Wire device as it comes with a unique 64-bit serial number. This number is used as an identification code for the transmitter, allowing the receiver to be paired with a specific transmitter only. We finally picked the type DS2438Z+ 1-Wire battery monitor IC with its integrated temperature sensor and the ability to run off supply voltages down to 2.4 V. With a maximum error of ±2 °C, at first blush the temperature measurement of the DS2438Z+ looks less accurate when compared to the DS18B20. In practice however the temperature measurements are quite acceptable.

The RF transmitter module is the previously mentioned type TXM-433-LR from Linx Technologies [3]. Internally it consists of a VCO locked by a frequency synthesizer taking its reference from a high-precision quartz crystal. The VCO output is amplified and buffered by an internal power amplifier is capable of outputting +10 dBm i.e. 10 mW into a 50-ohm load. At full power with a 50% transmitting duty cycle, the current consumption is around 5 mA which is low compared to similar RF modules. Except for an antenna, no external RF components are required. We used a helical-wound half-wave PCB antenna mounted perpendicular to the ground plane. The radiated power is adjustable using resistor R1. By default, a 0 Ω resistor is mounted for maximum output power but it may be required to reduce RF power in order to comply with local regulations. Please refer to the TXM-433-LR datasheet for more information.

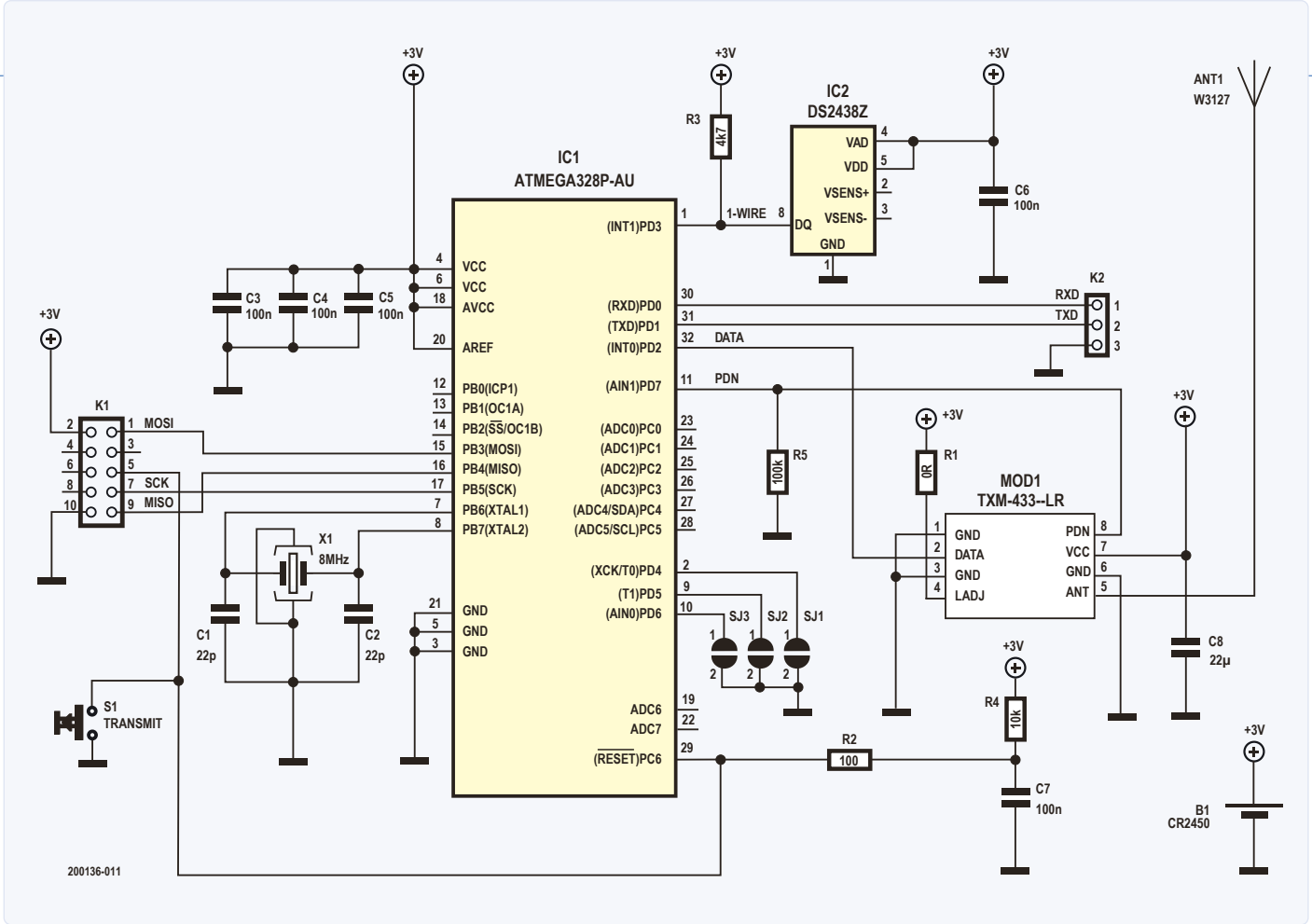


Figure 1: Temperature transmitter schematic. A type DS2438Z+ sensor emulates a DS18B20, allowing the circuit to operate down to 2.4 V.

The other components on the transmitter PCB include:

- a reset button;
- programming connector K1;
- (serial) debug connector K2;
- solder jumpers SJ1, SJ2, SJ3;
- one CR2450 battery to power the circuit.

The connectors K1 and K2 have no use during normal operation and are not mounted. The solder jumpers SJ1, SJ2, SJ3 allow the transmission interval to be adjusted between 2 minutes 8 seconds and 3 minutes 4 seconds. This reduces the risk of multiple transmitters sending data at the same moments resulting in data collision.

We went for a CR2450 lithium battery as it has more capacity (mAh) than the more commonly seen CR2032, at a fractionally higher cost (on eBay). With our prototype, the battery voltage was still 2.991 V after six months of continuous use. During a data transfer this was found to drop briefly to approximately 2.93 V. The transmission interval used was 2 minutes 24 seconds. Note that the battery lasts more than a year.

Transmitter — software

Most of the time the microcontroller is in Power-Down mode with the watchdog timer running. The watchdog timer is set to its maximum timeout of approximately 8 seconds. If the timeout period elapses, the watchdog timer generates an interrupt to awake the microcontroller. A counter in RAM is then decreased by one and if the value hasn't reached zero yet, the watchdog timer is restarted, and the microcontroller goes back into Power-Down mode. This process takes approximately 20 instructions.

If the counter reaches zero after a watchdog timer interrupt, the temperature registers and the 64-bit serial number of the DS2438Z+ are read. If this fails, the software retries reading the DS2438Z+ four additional times. The software also supports the DS18B20, the DS18S20 and the DS1822 temperature sensors. This can be useful if the transmitter is powered by a 3.3 V power supply instead of a lithium battery.

When the DS2438Z+ is present, the 1-Wire family code is changed to match the family code of a DS18B20. The CRC is also recalculated to match the new family code.

The 64-bit serial number with the new family code, temperature value and CRC are then transmitted two times via the RF module preceded by a preamble. The RF modules use OOK/ASK modulation meaning the carrier wave is turned on and off in concert with the data applied to the module. Please note that you can't simply feed serial data to the transmitter module and expect it to arrive flawlessly at the receiver. Without a 433 MHz carrier signal present, the receiver may pick up background noise or interference from other devices also operating at 433 MHz. To mitigate these phenomena, the data has to be encoded first to something that is more suitable for RF transmission. The 'preamble' allows the receiver to detect a known pattern and to synchronize with it before decoding the data. Among the more popular encoding standards are 'Manchester' [4] and the RadioHead library which is available for Arduino [5].

The data is not encrypted during transmission, as the primary goal is to measure the outdoor temperature which is usually not to be considered as sensitive information.

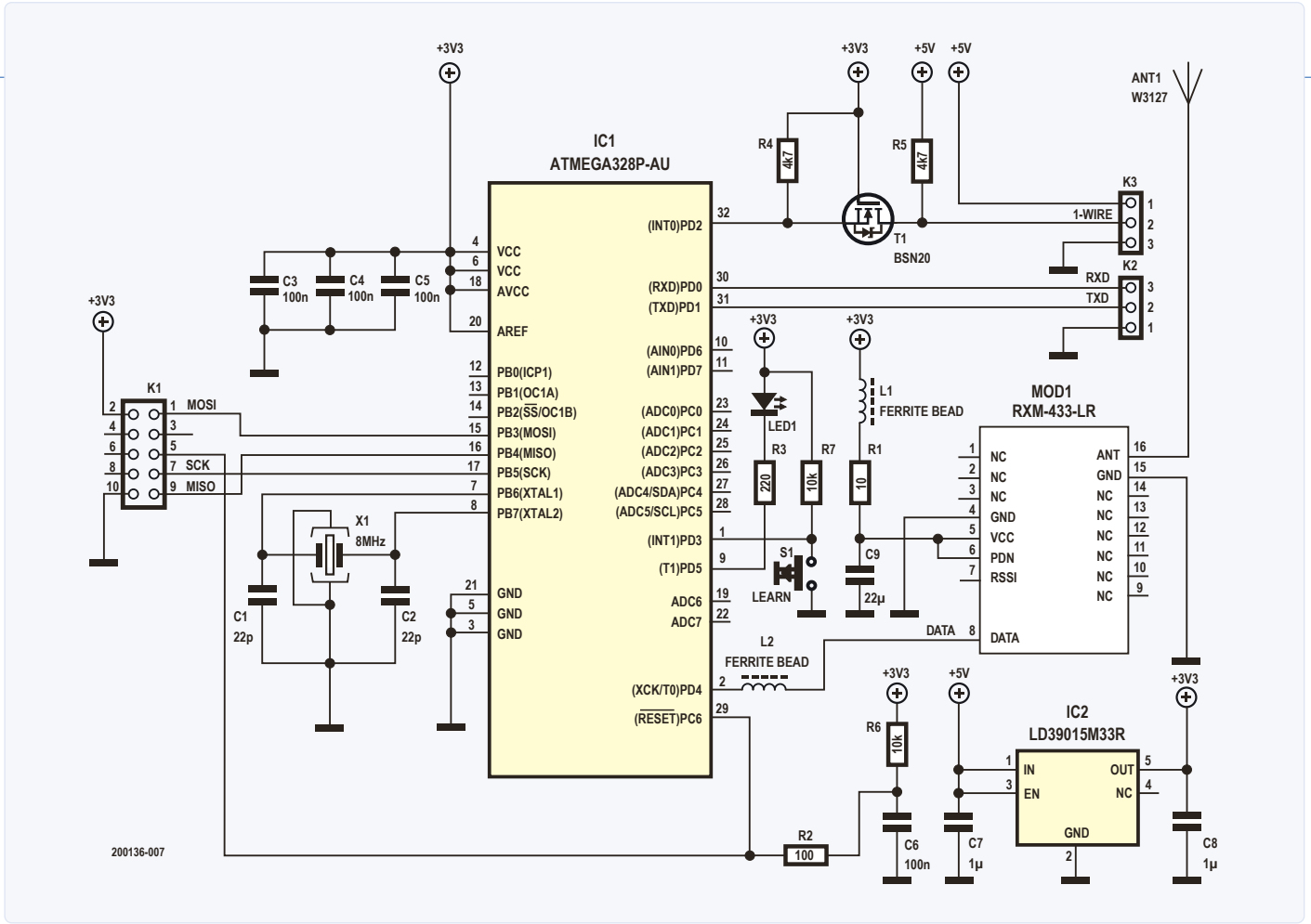


Figure 2: Temperature receiver schematic. The main tasks of the firmware running in the microcontroller include processing demodulated radio signals and interpreting 1-Wire commands.

After the data transmission, the status of the solder jumpers SJ1-SJ3 is read and the counter for the number of sleep cycles in RAM is set with a value of 16 plus the binary number represented by the solder jumpers (0-7, for a total number of sleep cycles between 16 and 23). The watchdog timer is then started, the microcontroller enters Power-Down mode once again and the cycle repeats.

After a reset, the whole cycle is also started. As such, pushing the reset button will initiate a wireless data transmission which is useful when pairing a receiver with a transmitter.

Receiver — hardware

The circuit diagram of the receiver shown in **Figure 2** is very similar to that of the transmitter and also based on an ATMEGA328P microcontroller. The only connections to the outside world are Vcc, GND and DQ (1-Wire data). The receiver shares the antenna with the transmitter. An RXM-433-LR receiver module from Linx Technologies is used [6]. This AM/OOK receiver module boasts "an advanced single-conversion superheterodyne architecture resulting in exceptional sensitivity and outstanding range performance."

COMPONENT LIST

Transmitter

Resistors

R1 = 0Ω, 1206 *
R2 = 100Ω, 1206
R3 = 4kΩ7, 1206
R4 = 10kΩ, 1206
R5 = 100kΩ, 1206

Capacitors

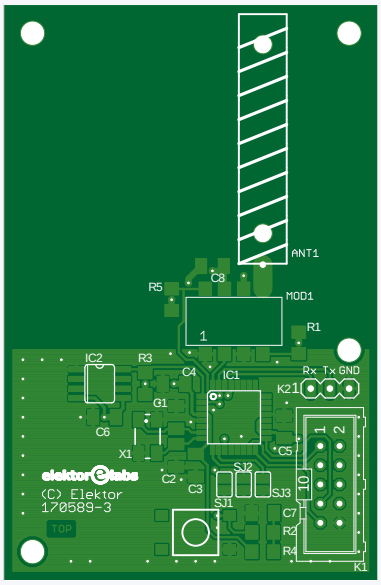
C1,C2 = 22pF, C0G/NP0, 1206
C3-C7 = 100nF, X7R, 1206
C8= 22µF, X5R, 10V, 1206, Kemet C1206C226M8PACTU

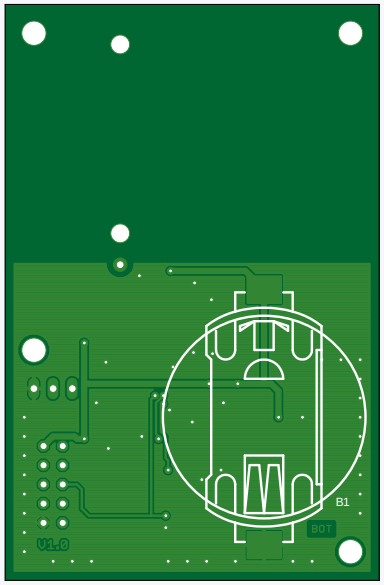
Semiconductors

IC1 = ATMEGA328P-AU, programmed
IC2 = DS2438Z+
MOD1 = TXM-433-LR, Linx Technologies (USA/Canada alternative: TXM-315-LR)

Miscellaneous

ANT1 = 433MHz helical PCB antenna, Pulse Electronics type W3127 (USA/Canada alternative: W3126)
B1 = CR2450 battery holder, Renata type SMTU 2450N-1-LF, w. CR2450 coin cell lithium battery





K1 = 10-pin (2x5) boxheader, 2.54mm pitch (optional)
K2 = 3-pin pinheader, vertical, 2.54mm pitch (optional)
S1 = tactile switch, SMT, TE Connectivity type FSM6JSMA
X1 = 8MHz quartz crystal, Abracon type ABMM2-8.000MHZ-E2-T or Würth Elektronik # 830055663

* see text

Note: a kit of parts for the transmitter is available from the Elektor Store.

COMPONENT LIST

Receiver

Resistors

R1 = 10Ω, 1206
R2 = 100Ω, 1206
R3 = 220Ω, 1206
R4,R5 = 4kΩ7, 1206
R6,R7 = 10kΩ, 1206

Capacitors

C1,C2 = 22pF, C0G/NP0, 1206
C3-C6 = 100nF, X7R, 1206
C7,C8 = 1µF, X7R, 1206
C9 = 22µF, X5R, 10V, 1206, Kemet C1206C226M8PACTU

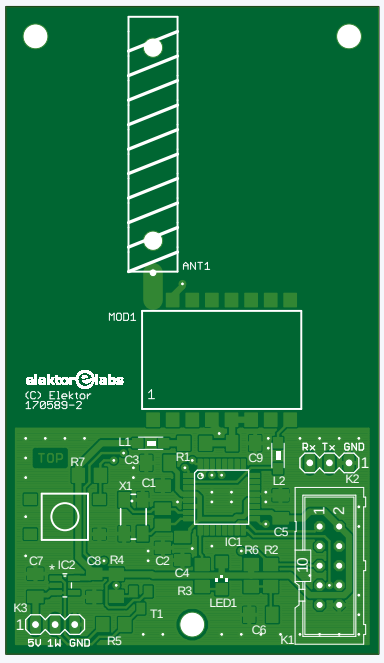
Semiconductors

LED1 = green, 1206, Broadcom Limited type HSMG-C150
IC1 = ATMEGA328P-AU, programmed
IC2 = LD39015M33R
MOD1 = RXM-433-LR, Linx Technologies (USA/Canada alternative: RXM-315-LR)
T1 = BSN20

Miscellaneous

ANT1 = 433MHz helical PCB antenna, Pulse Electronics type W3127 (USA/Canada alternative: W3126)
K1 = 10-pin (2x5) boxheader, 2.54mm pitch (optional)
K1 = 3-pin pinheader, vertical, 2.54mm pitch (optional)
K3 = 3-pin pinheader, angled, 2.54mm w. female terminal housing # 61900311621 and 3 pcs pre-crimped cable # 619100126015 (Würth Elektronik)
L1,L2 =ferrite bead, Fair-rite type 1206 2512067007Y3
S1 = tactile switch, SMT, TE Connectivity type FSM6JSMA
X1 = 8MHz quartz crystal, Abracon type ABMM2-8.000MHZ-E2-T or Würth Elektronik # 830055663

Note: a kit of parts for the receiver is available from the Elektor Store.



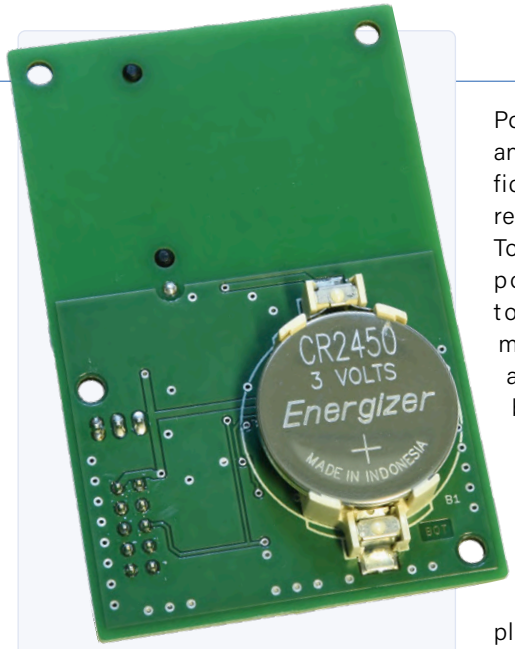


Figure 3: A type CR2450 3 V lithium coin cell is suggested for use in the transmitter. The '2450 is a bit dearer than the CR2032 but offers more capacity.

Power supply noise and ripple can significantly affect the receiver sensitivity. To ensure that the power delivered to the receiver module is as clean as possible, ferrite bead L1 and 10 Ω resistor R1 are connected in series with the Vcc pin of the module, and a 22 µF decoupling capacitor C9 was added. To avoid interference from the microcontroller, a ferrite bead L2 was also added to the data line.

Considering the RXM-433-LR has a 2.7–3.6 V supply voltage range, we opted to run the entire circuit off a 3.3 V supply voltage. The Nixie Bargraph Thermometer however operates at 5 V. To allow for an easy connection between the receiver circuit and the thermometer, an ultra-low drop, low noise voltage regulator type LD39015M33R (IC2) was added along with a bidirectional logic level convertor around MOSFET transistor T1.
When the receiver is used with a 3.3 V 1-wire bus as with the Elektor/Axiris IV-22 VFD Clock [7], the voltage 'behind' the regulator is still high enough for the circuit to work properly.

Besides a (not normally mounted) programming and a debugging connector, an LED and a pushbutton are connected to the microcontroller. Every time a wireless data packet is received, the LED changes status. This allows you to check if the transmitter is still functional. The button is used to pair the receiver with a transmitter.

Receiver — software

Until recently the 1-Wire bus from Dallas Semiconductors (now Maxim) was a very popular communications bus system to build MicroLAN networks for use in home automation systems, for example. Due to the increased use of wireless solutions, 1-Wire is obsoleting fast. Despite this, some 1-Wire devices such as the DS18B20 (and similar temperature sensors) are still widely used. They are cheap, accurate, easy to interface with a microcontroller and come in easy to solder TO-92 packages! These were the main reasons for using them in Elektor Nixie Bargraph Thermometer and IV-22 VFD clock projects, as well as to simulate a DS18B20 with our receiver module. As the original 1-Wire patents have expired, this is also quite legal to do even in the scorching heat of the MidWest.

The software listens to the RF receiver module as well as the 1-wire bus. When a data transmission arrives at the receiver, the encoded data is decoded and checked for validity. If a valid data packet is received, the received 1-Wire serial number is subsequently compared with the 1-Wire serial number stored in the microcontroller's EEPROM. When the two match, the scratchpad (containing among other things the temperature value bytes) of the simulated DS18B20 temperature sensor is updated, if not, the data packet is discarded.
The simulated 1-Wire slave supports the following 1-Wire commands (i.e. ROM and DS18B20 function commands):

- `Read Rom` [33h]
- `Match Rom` [55h]
- `Search Rom` [F0h]
- `Convert T` [44h]
- `Write Scratchpad` [4Eh]
- `Read Scratchpad` [BEh]
- `Copy Scratchpad` [48h]
- `Recall E2` [B8h]
- `Read Power Supply` [B4h]

Where 'Rom' of course equals: ROM, for sticklers. It is not possible to power the receiver module in parasitic mode. The `Read Power Supply` command will always return 'externally powered' status.

Please note that the 1-Wire slave functions of the receiver module have priority over the RF functions. If a 1-Wire bus communication is initiated while a data packet is being received and decoded, RF data reception is aborted. As both the thermometer and the VFD clock only read the temperature value once a minute, chances that this might happen regularly are relatively small. It's also why the shortest transmission interval of the transmitter is 2 minutes 8 seconds and not 2 minutes flat.

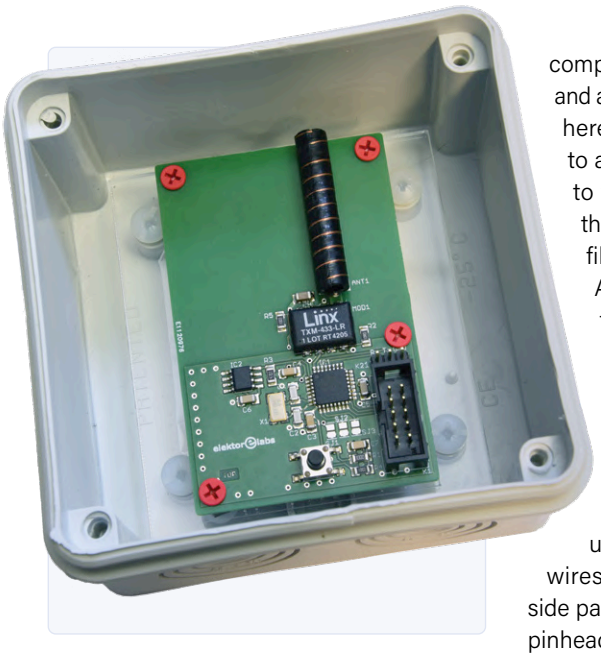


Figure 4: To permit outdoor operation, the transmitter was designed to fit in an electrical conduit junction box.

Construction and practical use

The transmitter and receiver module both come as preassembled and tested PCBs supplied through the Elektor Store. The component mounting plans for the PCBs and associated component lists are printed here for the sake of completeness and to allow a degree of open engineering to those wishing to collect the parts themselves. Likewise, the PCB Gerber files and the receiver and transmitter ATmega firmware can be downloaded from the article support page [8].

Some work is required to make the preassembled TX and RX boards function. Mount K3, a 3-pin 0.1-in. angled header and slide the three pre-crimped wires into the 3-way female connector from the backside until they snap into place. Guide the wires through the hole in the enclosure side panel and solder the other ends to the pinheader on the receiver board. Finish the connections with pieces of heat shrink tube. Make sure the GND, DQ and 5 V connections are right and match them on the thermometer. Now mount the receiver into its enclosure, connect it to the thermometer and provide power to the thermometer. The thermometer will now display full scale as the simulated DS18B20 has a reset temperature value of 85 °C, just like the 'real' DS18B20. If the receiver was never paired to a transmitter, the EEPROM may not contain a 1-Wire serial number. In that case, the 1-Wire slave functions are suspended, resulting in no display on the IN-9 tube.

Now put a CR2450 lithium coin cell into the battery holder on the rear side of the transmitter (**Figure 3**). Position the transmitter a few meters away from the receiver. Push the button on the receiver and the green LED will start to blink indicating the receiver is now in Learn mode. Now push the button on the transmitter and if everything goes well, the LED on the receiver will stop blinking, indicating that a good temperature value is received and that the receiver is now paired to the transmitter. As the thermometer reads the 1-Wire sensor once every minute, it may take some time before the actual temperature value is displayed. For outdoor use, mount the transmitter in a waterproof box together with a small bag of silica gel to keep it dry. A cheap solution is to use an electrical wiring/conduit junction box with an IP65 rating (**Figure 4**). Every time you turn on the thermometer, it may take a few minutes before the temperature value is received and displayed. During that time, the thermometer displays its full-scale value.

It is possible to use multiple transmitters and receivers at the same time. Since the transmitter changes the family code of the DS2438Z+ into that of a DS18B20, there is a minute chance that the serial number of the simulated DS18B20 is not unique when using one or more receivers in a 1-Wire network with other (real) DS18B20 temperature sensors present. Finally, in practical use we noticed that some Arduino Nano boards cause more interference (RFI) than others. This interference, along with building materials and metal structures in the path between TX and RX, may affect the effective range.

200136-01

SHOPPING LIST

- **Receiver module with revised acrylic scale and cable**
  www.elektor.com/170589-72
- **Transmitter module with battery and waterproof case**
  www.elektor.com/170589-73
- **Nixie Bargraph Thermometer (Kit) (original design)**
  www.elektor.com/nixie-bargraph-thermometer-170589-71
- **VFD-Tube Clock with ESP32 DevKit-C incl. acrylic case**
  www.elektor.com/vfd-tube-clock-with-esp32-devkit-c-170537-71

WEB LINKS

[1] **Nixie Bargraph Thermometer, Elektor Labs July & August 2018**: www.elektormagazine.com/magazine/elektor-201807/41730/

[2] **DS2438 Smart Battery Monitor — Datasheet**: https://datasheets.maximintegrated.com/en/ds/DS2438.pdf

[3] **Linx LR Series Transmitter Module Data Guide**: www.linxtechnologies.com/wp/wp-content/uploads/txm-fff-lr.pdf

[4] **Manchester Data Encoding for Radio Communications**: www.maximintegrated.com/en/design/technical-documents/app-notes/3/3435.html

[5] **RadioHead Packet Radio library for embedded microprocessors**: www.airspayce.com/mikem/arduino/RadioHead/

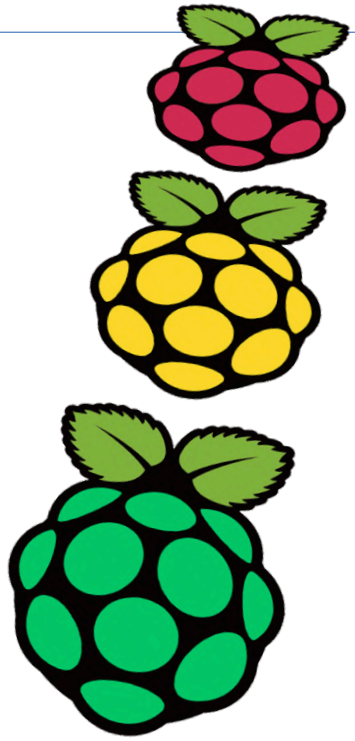
[6] **Linx LR Series Receiver Module Data Guide**: www.linxtechnologies.com/wp/wp-content/uploads/rxm-fff-lr.pdf

[7] **VFD-tube Clock with ESP32, Elektor Labs May & June 2018**: www.elektormagazine.com/magazine/elektor-201805/41547

[8] **Article resources and support page**: www.elektormagazine.com/200136-01

Multitasking with Raspberry Pi

Showcase: a traffic lights controller



By **Dogan Ibrahim** (United Kingdom)

Here is an example project reproduced from the newly published book *Multitasking with RPi*. The book is written for students, practising engineers, and hobbyists interested in developing multitasking projects using the Python 3 programming language on the Raspberry Pi computer.

Multitasking has become one of the most important topics in microcontroller-based systems, namely in automation applications. As the complexity of the projects grows, more functionality is demanded from the projects and such projects require the use of several inter-related tasks running on the same system and sharing the CPU (or multiple CPUs) to implement the required operations. As a result of this, the importance of multitasking operation in microcontroller-based applications has been steadily growing over the last few years. Many complex automation projects nowadays make use of some form of a multitasking kernel. In the book, the Python 3 programming language is used with the Raspberry Pi 4. Other models of Raspberry Pi can also be used without any change to the code.

The book is project-based. Its main aim is to teach the basic features of multitasking using Python on Raspberry Pi. Many fully tested projects are given in the book using the multitasking modules of Python. Each project is fully described and discussed in detail. Complete program listings are provided for each project. Readers should be able to use the projects as they are, or modify them to suit their own needs.

Showcase: a traffic lights controller

In this project, a simple traffic light controller is designed for a junction. The junction is located at the intersection of two roads: East Street and North Street. There are traffic lights at each end of the junction. There are pedestrian buttons located near the traffic lights on North Street. Pressing a pedestrian button turns all lights to red at the end of their cycles. A buzzer is then sounded to indicate that the pedestrians can cross the road safely. Also, an LCD is connected to the system to display whether the pedestrian cycle is running or the traffic cycle is running. **Figure 9.12** shows the layout of the equipment at the junction.

In this project, the following fixed times are given to each traffic light duration, and also to the duration of the pedestrian buzzer. For simplicity, both roads of the junction are assumed to have the same timings:

- Red time: 19 seconds
- Amber time: 2 seconds
- Green time: 15 seconds
- Amber+Red time: 2 seconds
- Pedestrian time: 10 seconds

The total cycle time of the lights in this example project is set to be 38 seconds. The sequence of traffic lights is assumed to be as follows (different countries may have different sequences):

Red ➞ Amber+Red ➞ Green Amber ➞ Green ➞ Amber ➞ Red ➞ Amber+Red ➞

Figure 9.13 shows the block diagram of the project.

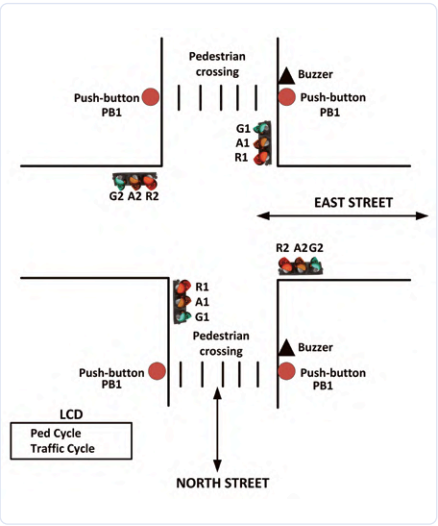


Figure 9.12: Layout of the equipment at the junction.

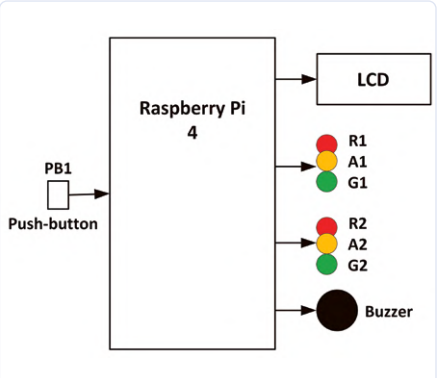


Figure 9.13: Block diagram of the project.

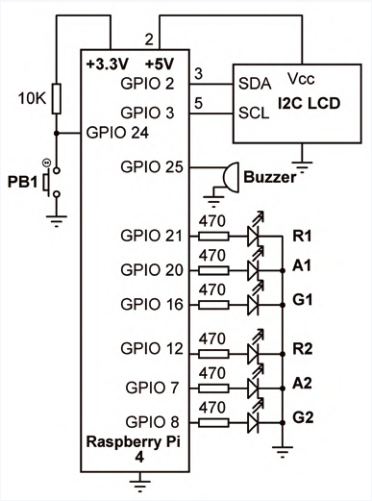


Figure 9.14: Circuit diagram of the project.

The circuit diagram of the project is shown in **Figure 9.14**. Red (R), Amber (A), and Green (G) LEDs are used in this project to represent the real traffic lights.
The following connections are made between the Raspberry Pi and road traffic equipment:

Raspberry Pi	Equipment
GPIO 21	LED R1
GPIO 20	LED A1
GPIO 16	LED G1
GPIO 12	LED R2
GPIO 7	LED A2
GPIO 8	LED G2
GPIO 25	Buzzer
GPIO 2	LCD SDA
GPIO 3	LCD SCL
GPIO 24	PB1 (push-button switch)

Figure 9.15 shows the full program listing (program name: traffic.py; download: [1]). At the beginning of the program the modules called *RPi*, *time*, *I²C LCD driver*, and *multiprocessing* are imported to the program and two queues named `pedq` and `lcdq` are created. Two functions are defined in the program with the names `ONOF` and `CONF_OUT`. Function `ONOF` has two arguments: `port`, and `state`. This function sends the state (0 or 1) to the specified GPIO port. Function `CONF_OUT` has one parameter called port. This function configures the specified GPIO port to output state. There are two processes in the program:

Figure 9.15: Program listing: traffic.py

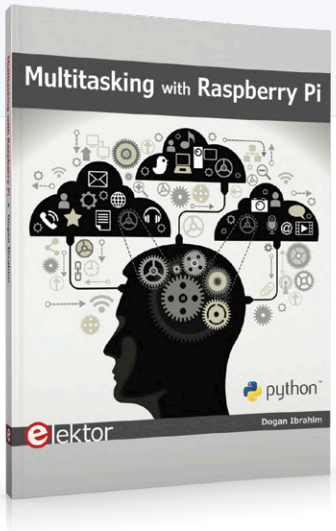
```
#-----------------------------------------------------------------------
#                    TRAFFIC LIGHTS CONTROLLER
#                    =========================
#
# This is a traffic lights controller project controlling lights
# at a junction. 6 LEDS are used to represent the traffic lights.
# Additionally a button is used for pedestrian crossing, and an
# LCD shows the state of the traffic lights at any time
#
# Author: Dogan Ibrahim
# File  : traffic.py
# Date  : May 2020
#-----------------------------------------------------------------------
import RPi.GPIO as GPIO                         # Import RPi
import multiprocessing                          # Import multiprocessing
import time                                     # Import time
import RPi_I2C_driver                           # I2C library
LCD = RPi_I2C_driver.lcd()                      # Import LCD
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)                          # GPIO mode BCM
pedq = multiprocessing.Queue()                  # Create queue
lcdq = multiprocessing.Queue()                  # Create queue

#
# This function sends data 'state (0 or 1)' to specified port
#
def ONOF(port, state):
   GPIO.output(port, state)

#
# This function configures the specified port as output
#
def CONF_OUT(port):
   GPIO.setup(port, GPIO.OUT)

#
# Process to control the lights
#
def Lights():                                   # Process Lights
   R1=21; A1=20; G1=16                          # LED connections
   R2=12; A2=7;  G2=8                           # LED conenctions
   Buzzer=25                                    # Buzzer connection
   CONF_OUT(R1); CONF_OUT(A1); CONF_OUT(G1)     # Configure
   CONF_OUT(R2); CONF_OUT(A2); CONF_OUT(G2)     # Configure
   CONF_OUT(Buzzer)                             # Configure
   ONOF(R1,0); ONOF(A1,0); ONOF(G1,0); ONOF(R2,0); ONOF(A2,0); ONOF(G2,0)
   ONOF(Buzzer, 0)

   RedDuration = 15
   GreenDuration = 15
   AmberDuration = 2

#
# Control the traffic light sequence
#
   while True:                                  # Do forever
      ONOF(R1,0); ONOF(A1,0); ONOF(G1,1); ONOF(R2,1); ONOF(A2,0);
   ONOF(G2,0)
      time.sleep(RedDuration)
      ONOF(G1,0); ONOF(A1,1)
      time.sleep(AmberDuration)
      ONOF(A1,0); ONOF(R1,1); ONOF(A2,1)
      time.sleep(AmberDuration)
      ONOF(A2,0); ONOF(R2,0); ONOF(G2,1)
      time.sleep(GreenDuration)
```

```
      ONOF(G2,0); ONOF(A2,1)
      time.sleep(AmberDuration)
      ONOF(A2,0); ONOF(A1,1); ONOF(R2,1)
      time.sleep(AmberDuration)

      while not pedq.empty():                # If ped request
         lcdq.put(1)
         ONOF(G1,0); ONOF(R1,1); ONOF(A1,0)  # Only RED ON
         ONOF(G2,0); ONOF(R2,1); ONOF(A2,0)  # Only RED ON
         d = pedq.get()                      # Clear ledq
         ONOF(Buzzer, 1)                     # Buzzer ON
         time.sleep(10)                      # Wait 10 secs
         ONOF(Buzzer, 0)                     # Buzzer OFF
         d = lcdq.get()                      # Clear lcdq


def Pedestrian():                            # Process Pedestrian
   PB1 = 24
   GPIO.setup(PB1, GPIO.IN)                  # PB1 is input

   while True:                               # Do forever
      while GPIO.input(PB1) == 1:            # PB1 not pressed
         pass
      pedq.put(1)                            # Send to Ped queue
      while GPIO.input(PB1) == 0:            # PB1 not released
         pass

#
# Create the processes
#
p = multiprocessing.Process(target = Lights, args = ())
q = multiprocessing.Process(target = Pedestrian, args = ())
p.start()
q.start()

#
# LCD Display control. Display 'Ped Cycle' or 'Traffic Cycle'
#
LCD.lcd_clear()                              # Clear LCD
LCD.lcd_display_string("TRAFFIC CONTROL", 1) # Heading

while True:                                  # DO forever
   if not lcdq.empty():
      LCD.lcd_display_string("Ped Cycle    ", 2)
   else:
      LCD.lcd_display_string("Traffic Cycle", 2)
   time.sleep(1)
```

Figure 9.16: Example display on the LCD.

SHOPPING LIST

- **Multitasking with RPi, book,** www.elektor.com/19357
- **Multitasking with RPi, e-book,** www.elektor.com/19360



`Lights`, and `Pedestrian`. At the beginning of the `Lights` process, the connections between the Raspberry Pi and the LEDs (traffic lights) and the Buzzer are defined and these ports are configured as outputs. Additionally, all these port outputs are cleared to 0 so that all LEDs and Buzzer are set OFF. Additionally, the LEDs are sequenced in the correct order with the correct timings. Towards the end of the function, it is checked as to whether the pedestrian button has been pressed. The pedestrian button is pressed if queue `pedq` is not empty. During the pedestrian cycle, the red lights are turned ON on both streets to stop the traffic flowing and give way to the pedestrians.

Also, the Buzzer is activated for 10 seconds during the pedestrian cycle to inform the pedestrians that it is safe to cross the road.

The Pedestrian process continuously monitors button PB1. If the button is pressed, then a 1 is sent to queue pedq so that process Lights can easily detect this action and start the pedestrian cycle.

The main program controls the LCD. When the program is started, the message 'TRAFFIC CONTROL' is displayed on the first row of the LCD. The second row of the LCD continuously checks queue `lcdq` and displays either 'Ped Cycle' or 'Traffic Cycle'.

An example display on the LCD is shown in **Figure 9.16**.

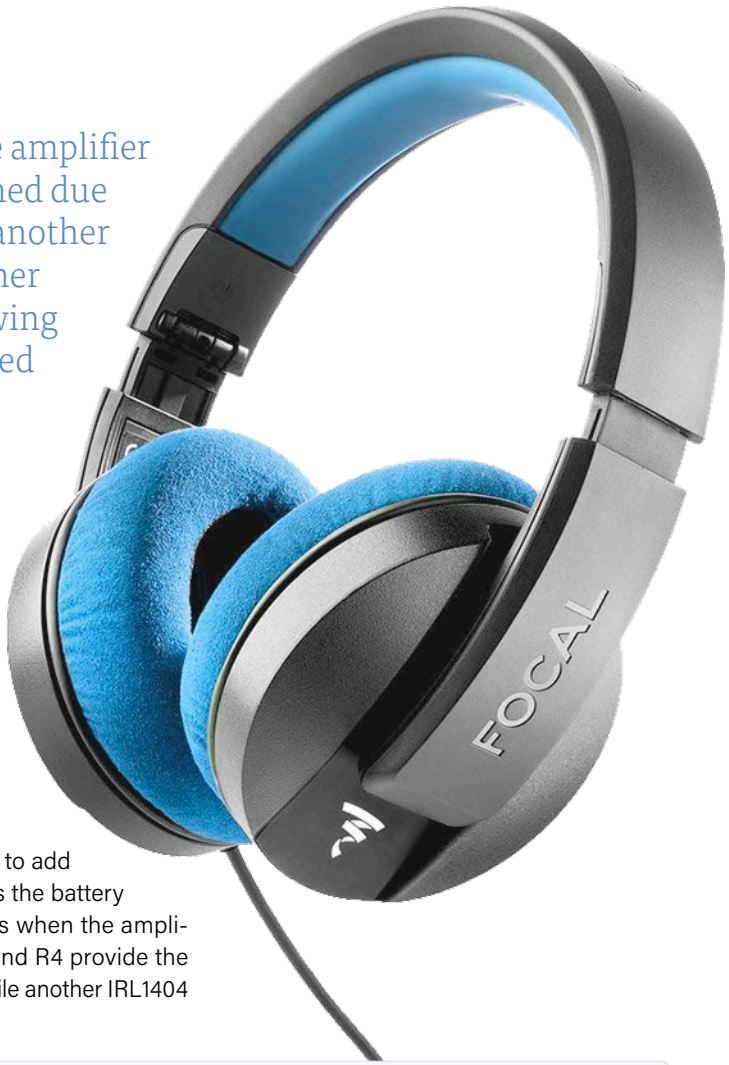
200381-01

WEB LINK

[1] **Traffic.py program download**: www.elektormagazine.com/200381-01

Timer for Headphone Amplifier

By **Glyn James**



Having built a battery-powered valve headphone amplifier I discovered that the battery was often fully drained due to forgetting to switch it off. After replacing yet another empty battery, I decided it was time to build a timer to shut down the amplifier automatically. Reviewing my standard usage, a period of 90 minutes seemed to provide the optimal target duration. A circuit based upon an oscillator was quickly discounted due to the potential for audible interference and poor power consumption. Instead, a resistor-capacitor (RC) network together with a Schmitt trigger and MOSFET provided precisely the function required with very little current drain.

The circuit **(Figure 1)** is built around a 4093 quad NAND IC with Schmitt trigger (IC1). This device can withstand supply voltages of up to 15 V, more than enough for my purposes. A pair of the gates (IC1a/b) is used to drive the gate of an IRL1404 power MOSFET (Q1) to which the headphone amplifier is connected in the high side as the load (CN2). On the input side of the NAND gates are a capacitor (C1) and a resistor (R3) whose values allowed my amplifier to remain on for about 100 minutes using a 12 V battery. The switch is a sprung momentary ON-OFF-ON SPDT type (SW1). This was used in preference to two momentary push switches that could short the battery should they be pressed simultaneously.

Momentarily pressing the switch in the 'on' direction (towards R2) charges C1 via R2 and turns the amplifier on. C1 eventually discharges via R3 at which point Q1 and the amplifier are switched off. Briefly pressing SW1 in the opposite direction (towards R1) quickly discharges C1 via R1 providing a manual method to switch the amplifier off.

Not wanting to waste the remaining NAND gates (IC1c/d) I decided to add a voltmeter that displays the battery status for 2 to 3 seconds when the amplifier is switched on. C2 and R4 provide the timing for this feature while another IRL1404

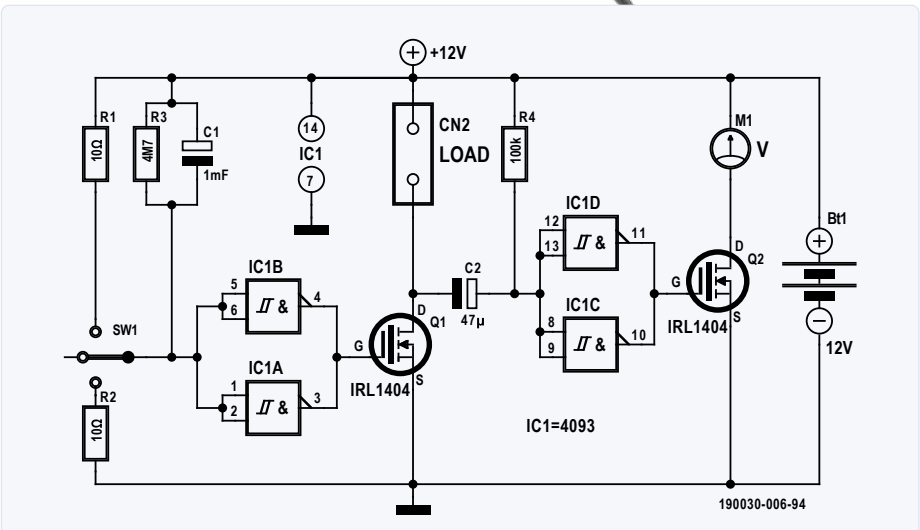


Figure 1: As shown, the on-time for the timer is around 100 minutes when powered from a 12 V battery. The secondary timer displays the battery voltage for around 3 seconds.

MOSFET (Q2) is used to power the voltmeter. As Q1 is turned on, C2 charges through R4, enabling the voltmeter via Q2. As soon as C2 is fully charged, the voltmeter turns off again. C2 discharges through the load (my amplifier) and R4 when the load is powered off.

The voltmeter (**Figure 2**) is a miniature seven-segment LED type that is commonly available at low cost from many online shops. Various types were tested and found to have a current draw of only 6 to 16 mA, so the brief display of the voltage has minimal impact on battery life. The meter chosen featured a circular black plastic bezel that made it easy to mount in a panel.

The MOSFET selected can easily handle any likely current requirements of battery circuits as, according to the data sheet, it can handle currents of up to 100 A. They also handle gate-to-source voltages of up to 20 V, with the gate threshold voltage ($V_{GS(th)}$) lying between 1.0 and 3.0 V. The Schmitt trigger in the NAND gates ensure that both MOSFETs are either fully switched-on or -off, guaranteeing minimal current consumption. My multimeter can measure currents down to 100 nA, but testing indicated that the current draw when off was below this level.

The drain-to-source on-resistance of the MOSFETs is less than 6 mΩ, meaning the circuit has little impact on the amplifier and that the voltmeter measurement reflects the true battery voltage. Tests with the MOSFETs showed no warming at currents of up to 800 mA, so no heatsink was deemed necessary. Should larger loads be connected to the circuit, the need for a heatsink should be reviewed. I didn't see any need for a flyback diode for my application but, if you are powering an inductive load, a 1N4007 could be reversely fitted to protect against back-EMF between the drain and source of Q1.

190030-01

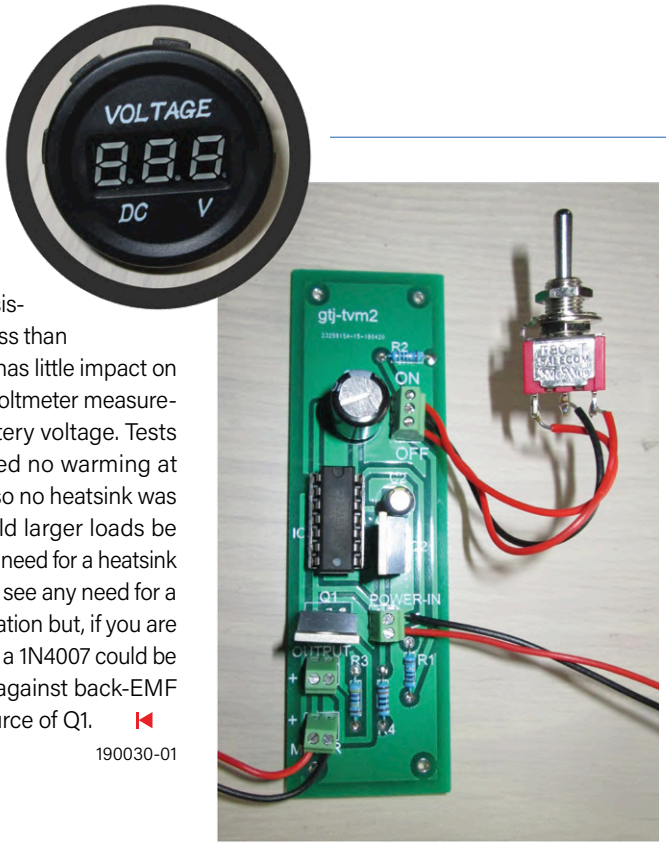


Figure 2: The final circuit built on a PCB with the momentary ON-OFF-ON switch and the selected panel-mounted voltmeter.

Open-Network Weather Station Mk.2

Part 2: Software

By **Mathias Claußen** and **Luc Lemmens** (Elektor Labs)

In the May & June 2020 issue [1] we presented our new open-source weather station designed here in the Elektor lab. The system consists of a mechanical kit (wind sensors, rain sensors, and housing) as well as the necessary electronics built around an ESP32, all available from the Elektor Store. It reads measurement output values from the sensors and sends them to cloud services such as openSenseMap and ThingSpeak. Hardware is, of course, only part of the project and in this article you will learn how the function blocks of the modular software interact. Thanks to an intelligent Mapper, sensor data can be flexibly linked to communication channels. The web server, which plays a central role in the project, will also be examined in detail in this article.

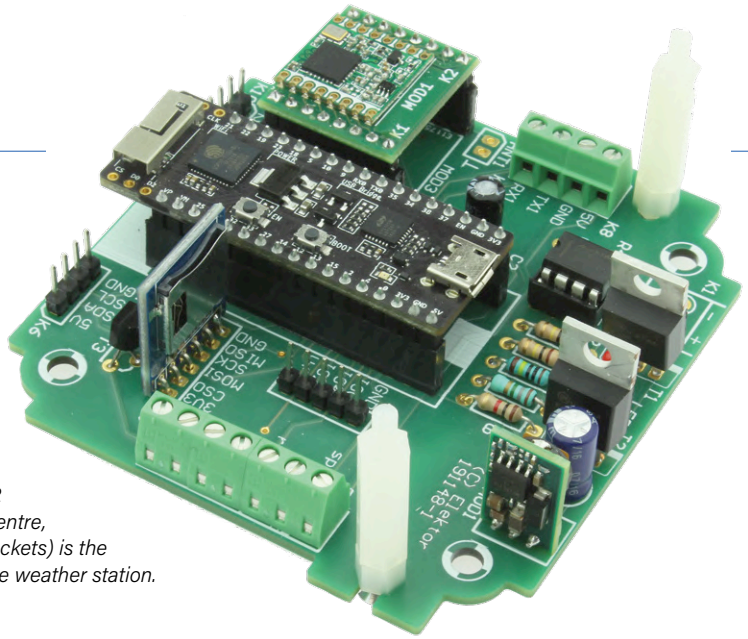


Figure 1: An ESP32 controller (in the centre, mounted on pin sockets) is the beating heart of the weather station.

Here in the Elektor lab we are always happy when we get reader feedback on any of our projects that suggest how they can be improved and expanded. With this in mind, we always design the software from the outset in a way that allows extensions to be implemented easily. Continuing on this theme, you will also note that our weather station [1] hardware can also be easily expanded. With its I/O Matrix, the ESP32 (**Figure 1**) provides flexible options for implementing the required functions on almost any of its interfaces. This applies regardless of whether it is SPI, I²C, or just a simple digital input for reading the status of a push button.

Let's have a look at the software modules and find out why the design is so flexible. We will start with the functions that need to be implemented in software [2] [3].

The basic functions

Sensors

The sensors supply measurement values for the three basic parameters of wind direction, wind speed, and rainfall. They communicate over an interface implemented with GPIO pins. An I²C bus is also available and this — amongst other things — controls and reads out temperature, humidity and air pressure values from the BME280 sensor. The following sensor types are also supported by the latest version of the software:

- VEML6070 (UVA)
- VEML6075 (UVA/UVB)
- TSL2561 (luminosity)
- TSL2591 (luminosity)
- WSEN-PADS (barometric pressure)

For air quality and measurement of fine particulate concentration, the software can address an SDS011 or Honeywell HPMA115SXXX sensor via a UART interface.

Other sensors can also be added to the system but would require modification of the firmware to support them.

Upon booting, the firmware will search automatically for any connected sensors and attempt to select an appropriate driver.

Cloud services

Cloud services provide a method to store measurement data from the weather station at regular intervals, provided there is a Wi-Fi link available. Supported services include ThingSpeak and openSenseMap, as well as a connection via an MQTT broker to the local cloud. If you use an MQTT broker, such as Mosquitto and a Raspberry Pi, you could use Node-RED [4] to process and save the data.

As well as defining which data values will be transmitted to these services, the time interval between uploads can also be configured.

SD card, display and pushbutton

An SD card slot is provided in the weather station so that measurement data can be stored when there is no active Wi-Fi connection. The slot accepts cards with a capacity greater than 4 GB. Measurement values from all the sensors are saved on the SD card (in CSV format) at user-definable intervals. The data is written directly into the main directory, with a new file being created each day.

The station display is used to show status messages (**Figure 2**). After booting, the Wi-Fi status is shown on the display and an indication is given of whether the SD card is currently in use by the software or whether it can be safely removed.

When the display is active, a press on the pushbutton allows the SD card to be unmounted or mounted. If the button is not pressed for 30 seconds, the display switches off to save energy. Pressing the button again turns the display back on. In addition to displaying the SD card status, it also shows the Wi-Fi network status and received signal strength. The station's IP address is also displayed.

An interesting feature of this push button is that is connected to the same pin as the ESP32 boot button. After the firmware boot-up period, this pin becomes freely available for use by the user software (see below).

Web server and web pages

As with many Elektor projects, a web server with configuration options is used. For example, in **Figure 3** you can see the setting of the time and date. In order for the data to

Figure 2: The display shows status messages. 'NOSD' indicates that either no memory card is inserted, or it is not recognised.

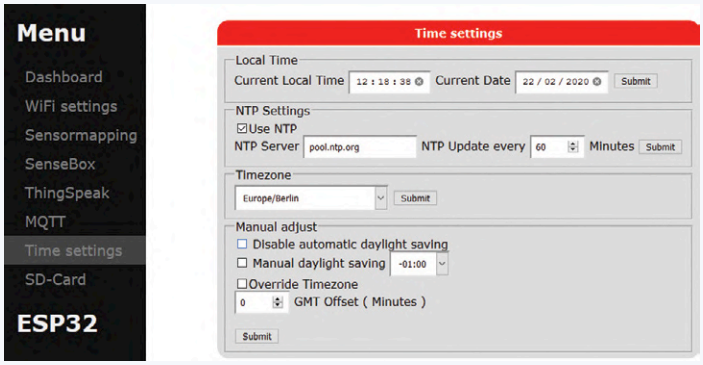


Figure 3: One of the central elements of the software is the web server, which supplies the web pages so a user can configure the weather station.

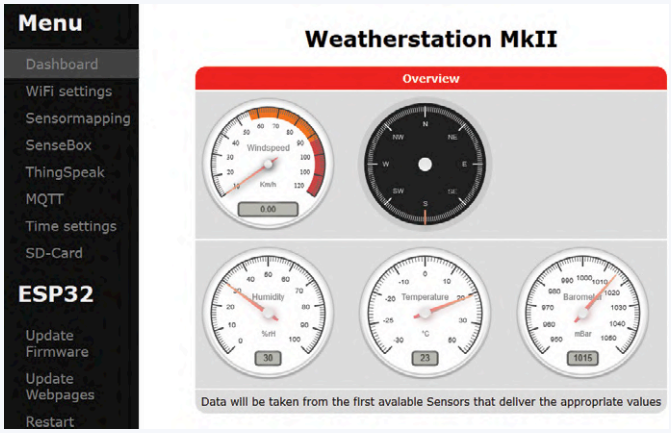


Figure 4: The web server also provides the most important values measured in a browser (for example, on a smartphone) that are then displayed as graphics.

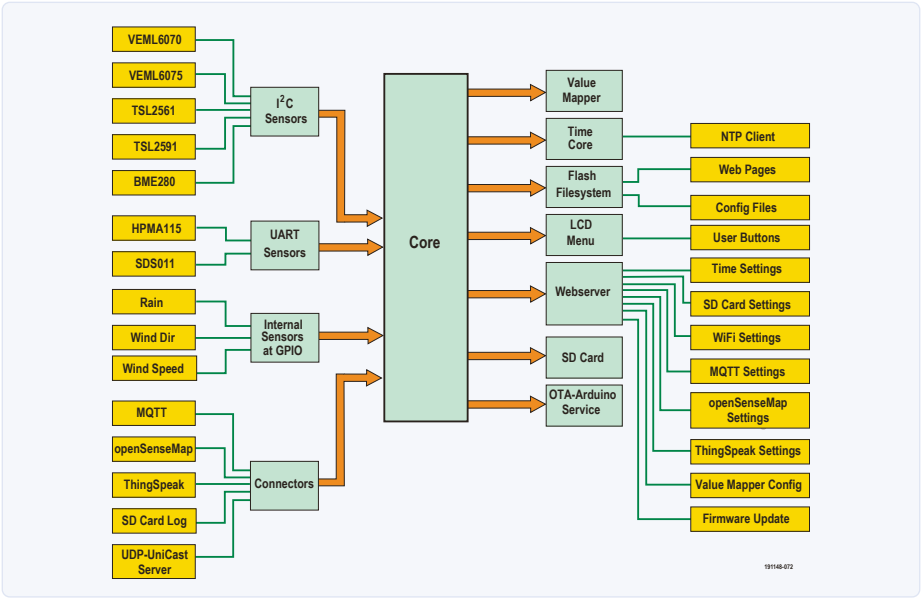


Figure 5: General overview of the software modules.

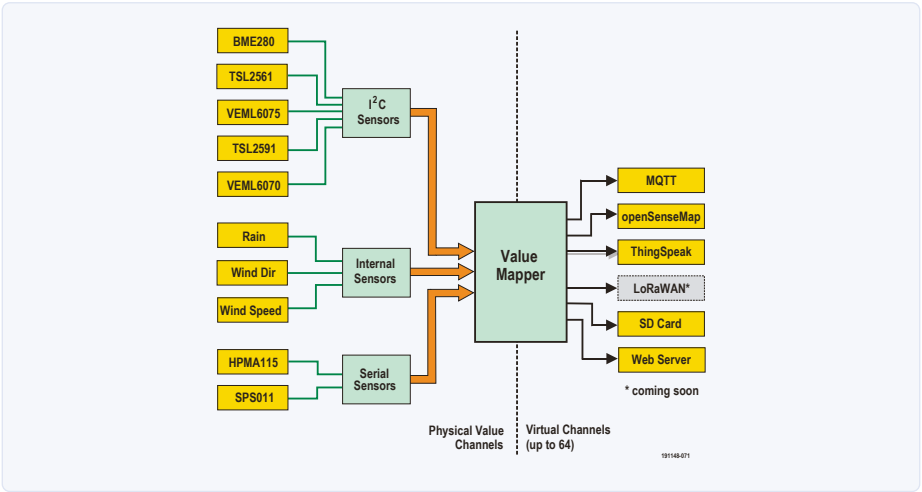


Figure 6: The Mapper connects the data sources (sensors) with the data sinks (Connectors to the cloud platform and the web server for display purposes). It provides the system with 64 virtual measurement channels.

be displayed in a browser, several interlocking components are required. Our ESP32 web server not only delivers web pages displayed in a browser, it also sends data to the browser on demand. JSON is used so that the data can be easily parsed by the web browser. The web pages not only consist of HTML but also use several lines of JavaScript.

This means that the computing power required to present the display can be partially off-loaded from the ESP32 to the web browser. One example is the dashboard (**Figure 4**) showing the weather station readings that can be displayed in a browser of your choice (on a smartphone, for example). The output is not transferred as graphics files but is instead generated by JavaScript in the browser.

Internals of the ESP32 firmware

All of the functions described will of course need to be supported by software. We will start with an overview of the individual parts and see how they work together to produce a functioning weather station. An overview of the software modules is shown in **Figure 5**.

First, we will look at how the measurement values from the sensors are processed. There are three main parts here: firstly we have the drivers that address the sensors; then come the Connectors that send data from the station to the cloud services and also write the data to the SD card; and finally, a Mapper that functions as an 'intermediary'.

Mapper

The Mapper provides 64 virtual measurement channels in the system that can be queried by the Connectors. A virtual channel in the Mapper can be assigned to the measurement value from a sensor. It should be noted that some sensors can output several different measurement values. The BME280, for example, generates three: air temperature, humidity and pressure. If one of the Connectors now wants the value for virtual channel 0, the Mapper carries out a translation according to an internal table, fetching the corresponding value via the appropriate sensor driver. The relationships are illustrated in **Figure 6**.

This may sound a bit complicated at first. A big advantage is that the connector does not need to know how to address the sensor. All of this is hidden by the Mapper. If another bus or other sensor type is added later, only the Mapper and drivers need to be modified. The connectors themselves continue to work as before.

The Mapper is also connected to the web server. This not only queries the measured values of the individual virtual channels

TASK PROGRAMMING

The connectors are implemented in the software as separate tasks so that they can read and process sensor data independently of one another. This creates a few problems that need to be solved. One of these is possible simultaneous access to the Mapper and thus the sensors on the bus. Take, for example, a case where two Connectors, Tom and Jerry (you know already this will end in a fight), are both actively fetching data from the Mapper every minute and thereby ultimately accessing the sensors. Tom might start reading data first and then need to wait until the most recent measured values are available or have been converted and so relinquishes computing time to the processor to allow another task to be active. Now Jerry also needs to get data from the Mapper, possibly from the same sensor connected to the bus that Tom is accessing. If an I²C bus is involved, it could be that Tom has just initiated a data request to a sensor and is waiting for the sensor to complete the process. If Jerry also starts reading data on the I²C bus, especially if it's from the same sensor, communication will become corrupted.

In the operating system of the ESP32, FreeRTOS, there are several mechanisms that can be used to solve this problem. The simplest method is to use a mutex (mutual exclusion). A mutex can be thought of as a single key that allows access to a restricted area. If you want access to this area you firstly need to get the key (mutex) to gain entry. The area remains locked to other users after you enter. Other users wanting access to this area must then wait until you are finished and have replaced the key. Only then can the next person acquire the key. With FreeRTOS, the developer must first ensure that the area is protected by a mutex and that the key (i.e. the mutex) is replaced after leaving the protected area. If this does not happen (as occurs more often than developers would like), the system grinds to a halt because the key has been 'misplaced.' This deals with the first point of how to regulate access to data. We also need to (re)allocate processing time when a particular task has no work to do currently. Software without an operating system generally runs all the tasks in a 'superloop' and it's necessary to devise mechanisms to allocate processing time to the tasks. When using an RTOS, a task can be put to sleep so that the OS can select other tasks that are not sleeping and execute them. When working with Connectors (e.g. the one for openSenseMap), the time between two send intervals is not spent in a loop using a simple `delay()` function but uses another mechanism (very similar to a mutex) called a binary semaphore. A mutex and a binary semaphore have a very similar function in FreeRTOS with only small differences. More about this can be found in our FreeRTOS article series [5]. The waiting time for a semaphore can be specified within FreeRTOS. If a semaphore becomes free during the waiting time, an appropriate response is received from the operating system. In this manner a defined waiting time or a defined interval can be created and computing time can be automatically assigned. But why use a semaphore and not simply tell the operating system with the instruction `vTaskDelay()` how long the task should continue to run? The answer lies in the method used to make changes to the configuration settings that the user can set via the web interface. If there is a time-out while waiting for a semaphore, the configuration will not be changed. If there are no errors flagged while waiting for a semaphore, the new configuration data can be applied. The configuration is then completely reloaded and applied so that the waiting process can begin again.

in order to display the data itself, but also requires information about connected, supported and missing sensors.
Each sensor value has a defined address within the software. This consists of the bus, the value type, and an ID in order to identify the sensor required for each access. For example, *'BME280.0.Pressure.I2C'* can be broken down to: *BME280*, which is the sensor identifier (good for code readability); *0*, which is the ID of the sensor value; *Pressure*, which relates to the measured parameter; and *I2C*, indicating that the sensor communicates via the I²C bus of the ESP32. From this description the Mapper knows that it is an I²C device and that the appropriate driver for access must be selected.

Driver

The Driver receives information from the Mapper about the type of value and the ID of the value. From this it determines which sensor is to be addressed and which value has to be fetched from the sensor.
On the I²C bus this is implemented by the I²C driver searching the bus for all known sensors after the booting is complete. This is undertaken by issuing a *START* on the bus and checking whether a device responds. If a device responds, it is marked as available and a *STOP* is sent. This sequence is repeated for all known sensors. Upon completion, a list is created indicating which devices are connected and active on the bus. Using this list, the Driver can ascertain whether an attempt to access a sensor will immediately generate an error (sensor not present), or whether an attempt should made to access the sensor. If a value is successfully read from the sensor it is returned to the Mapper. Otherwise, a suitable error response is returned.

Connectors

A software Connector provides the link between the measurement values and the location where that data is to be stored (the data sink). The Connectors are designed as separate tasks within the ESP32 firmware so that they can read and process sensor data independently of one another. Each Connector basically has a list of virtual channels that must be read from and processed so that they are in a compatible format to be used by other applications, such as the ThingSpeak IoT analytics platform.
Due to task programming approach, the Connectors are independent of each other and this can cause a few problems that need to be addressed (see **'Task programming' box**).

The web interface

Now we know how data is acquired from the sensors and passed to the cloud services, we

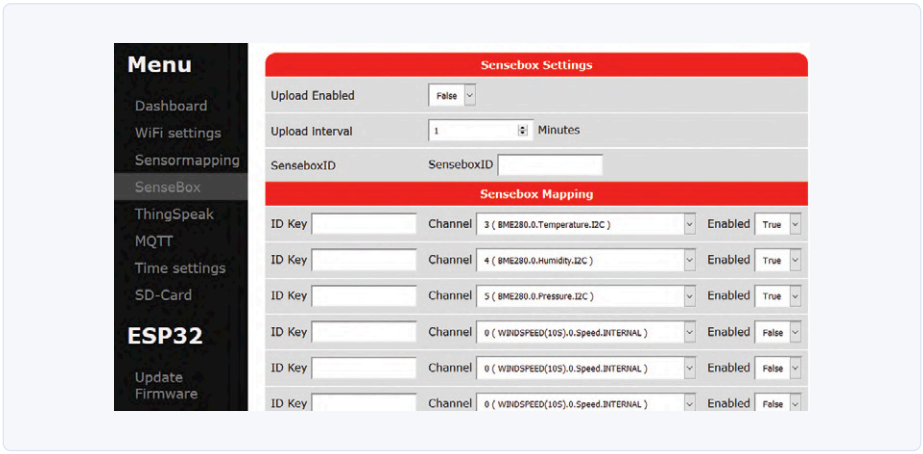


Figure 7. Configuring access to openSenseMap, …

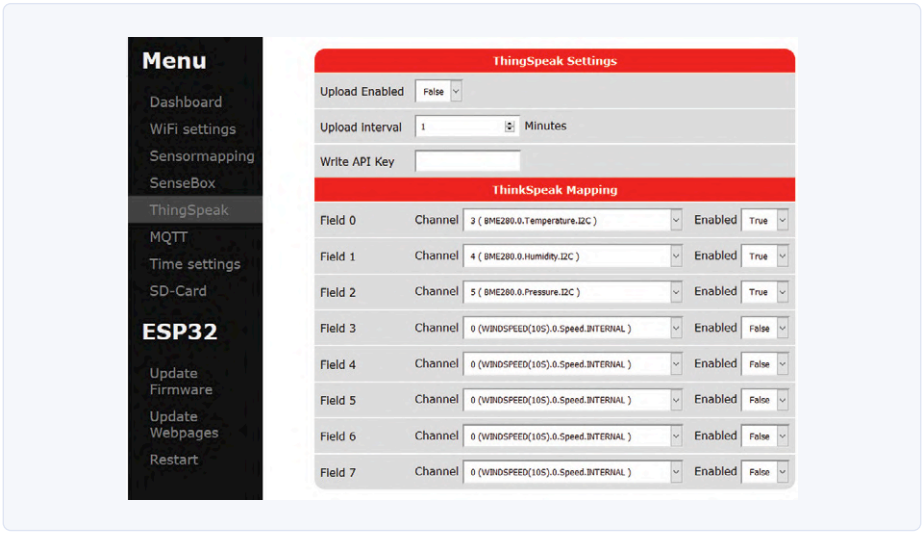


Figure 8: … on ThingSpeak and …

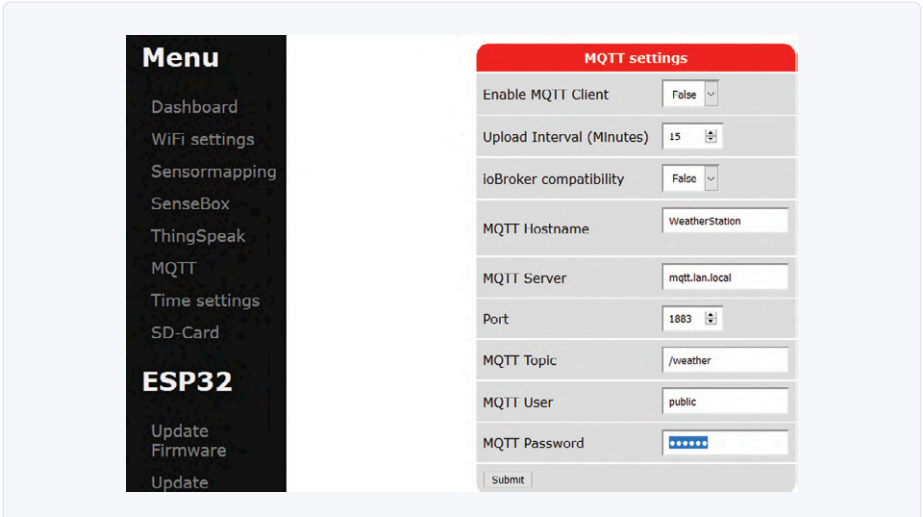


Figure 9: … to the MQTT broker. Each configuration page uses different control elements.

can look at the weather station user interface. The web interface itself is divided into several parts. For the Mapper and each Connector there is a separate modular section in the web interface so that new Connectors can be connected to the existing web server quite easily.

The web server provides the web pages and data that the browser needs to be able to render the web interface. The files for this are in part of the ESP32 flash memory known as SPIFFS. So far, most readers will probably have only used this memory in their own programs for read operations in order to output web page information stored in the ESP32. However, like an SD card, files such as JSON files can also be written to the flash memory using SPIFFS that can then be read directly by the web server. This method, however, has its disadvantages. Flash memory only offers a limited number of write cycles. Therefore, you should not rely on the flash memory to store files, such as Log files, that are regularly updated. When this memory fails, you can't replace it as easily as you can an SD card.

The firmware uses some of the SPIFFS memory to store part of the configuration files, mainly for the Connectors. **Figures 7**, **8** and **9** show how the user can configure the Connectors for the ThingSpeak and openSenseMap cloud platforms and the MQTT client. It can be seen that these pages use different control elements. The pages must be generated dynamically depending on the Connector. Here in the Elektor lab we have already used such a mechanism for other ESP32 web server applications. The function carried out depends on the URL that the web browser sends to the web server. This is then used to transfer configuration information and parameters from the system to the web server.

JavaScript

It is not only the web server that plays a part in building the web interface but also the web browser. Some of the web server work can be off-loaded to the browser, which reduces the workload for the ESP32. Our web page makes use of this when rendering the dials for the dashboard display. Instead of transferring them as graphics files they are drawn by the browser using scripts (JavaScript) at runtime. The preparation of data and tables are also not carried out in the ESP32 but by scripts in the web page. To make this possible, the ESP32 must create a path so that the web page in the browser has access to the data.

ESP32 AND JAVASCRIPT

Modern browsers can download several files simultaneously and, depending on the browser, there can be 4, 6, 8 or even more requests in parallel. In order to reduce the time that the user needs to wait while the page loads, the browser will start to execute the scripts to display measurement values, tables and display elements on the page. If the scripts are distributed over several files in order to make the code a little clearer to the programmer, it may be that parts of the code required have not yet been loaded. Thus, their execution will terminate and throw an error. To avoid the problem of partially loaded code, a script is used to load all the code into a single file. This problem occurs with the ESP32 because the web server running on the development system is able to deliver the data fast enough.

The other hurdle is loading multiple files within JavaScript. These are carried out concurrently with script execution and signals via a callback to indicate whether the file has been loaded. If several files are required at a single moment in time, the loading needs to be linked (i.e. allow File1 to load and then give a callback to load File2, whereupon File3 is then loaded). To avoid this, a small helper section was written that provides an array of URLs and memory for the replies, thus automating this task for us. Inside the file basic.js you can see how the function `loadMultipleData` works.

Interactions between the web interface, web server, and firmware are something that should not be underestimated in your own projects. Stumbling blocks lurk not only in the firmware, but also in the browsers that run on different devices. Every browser behaves a little differently and may need a certain amount of tweaking of the scripts so that the page is properly displayed.

This is where dynamically generated files can play an important role. The following dynamic files can be requested by the web browser:

/setWiFiSettings
/getSSIDList
/mapping/mappingdata.json
/devices/supportedsensors.json
/devices/connectedsensors.json
/mapping/{}/value
/mqtt/settings
/sdlog/settings.json
/sdlog/sd/status
/sensebox/settings.json
/sensebox/mapping/{}
/sensebox/mapping.json
/thingspeak/settings.json
/thingspeak/mapping/{}
/thingspeak/mapping.json
/timesettings

The response is always a JSON file allowing JavaScript running in the browser to parse and process its content very easily.

Curly brackets *{}* are used to indicate a parameter for a dynamic function. For example */mapping/0/value* returns the sensor value from the first logical channel and */mapping/1/value* the value of the second logical channel. */sensebox/mapping/0* returns the value from the first logical channel that is transferred to openSenseMap. A look at the source code shows how the whole thing is implemented. One example is the command that assigns the function in the web server:

```
server->on("/sensebox/
   mapping/{}", HTTP_
   GET, GetSenseboxChMapping );
```

This tells the web server that when accessing */sensebox/mapping/* the last part of the URL is to be seen as a parameter and the function `GetSenseboxChMapping` has to be called. In the function `String IDs = server->pathArg(0);` this parameter is processed as a string.

A problem that can occur when using the ESP32 together with JavaScript is described in the **'ESP32 and JavaScript' box**.

System start up

A simple LCD can be useful for displaying data in such projects but usually requires some buttons for input operations. In this case, the number of buttons is limited to one to reduce the number of holes required in the weather station's housing. Let's take a look at how the display functions and how the Boot push-button is used here. By holding the push-button down during power-up, the ESP32 is put into bootloader mode. During normal operation this push-button then becomes freely available for the software to use for other purposes. On power-up with the push-button held down, the station switches to AP mode (an appropriate message is also shown in the display). It displays a new Wi-Fi network with the name *ESP32-xx-xx-xx*, which can be connected to a device. If the connection is successful, the weather station configuration page can be called up at the URL *http://192.168.4.1.* The layout of the page should be familiar as it has been used in other projects. Once the Wi-Fi parameters of your local network have been entered, the ESP32 restarts and should then connect to the Wi-Fi network. In order to see how strong the signal is, the RSSI value is also displayed.

The LCD is controlled using the same push-button. To save energy, the display switches off after 30 seconds. To reactivate it, the push-button needs to be pressed briefly once. This action activates the backlight and the status information is displayed again. If the display is active, pressing the button again allows an SD card to be mounted, or a mounted card to be unmounted. If the first line of the display shows '_SD_' it is indicating that the SD card has been successfully mounted. 'NOSD' is displayed if the card has not been mounted or was not recognised.

Push-button contact de-bouncing is taken care of in software. Pressing the button generates an interrupt on both the rising and falling edge of the input signal (this is due to the architecture of the ESP32 and its GPIO controller). The interrupt routine must determine whether the interrupt is a result of a rising or falling edge. De-bouncing is taken care of by measuring the time between pressing the button and releasing it. If this is less than 100 ms, the key press is ignored. When a valid key press is detected, the interrupt routine creates a semaphore so that a separate task is then able to query it.

The LCD control is implemented in a similar way. A separate task updates the display at one second intervals for a period of 30 seconds, then goes to sleep and waits for the key to be pressed.

Settings

Once the firmware has been installed, the weather station creates a basic configuration assuming a rain gauge, anemometer and BME280 sensor are connected. Should the BME280 not be connected, this will not

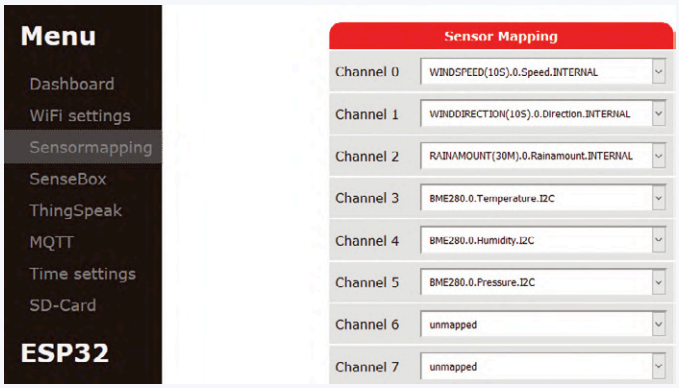


Figure 10: Assignment of sensor values to virtual channels.

be a problem; the corresponding channel will simply not output any value.
During setup, the virtual channels must first be configured by assigning or mapping sensor values to virtual channels (see **Figure 10**). These channels are later queried by the software Connectors that implement the connection to data sinks such as openSenseMap, ThingSpeak or the SD card.
The settings themselves also have an impact on the dashboard. From the virtual channels it acquires the values for wind direction, wind speed, temperature, humidity and air pressure, specifically the first of these values that can be found in the virtual channels.
The software connectors are not just used in the link between the weather station and the cloud services. Saving data to the SD card is also regulated by a Connector. For the SD card the settings are quite simple and consist of the interval at which the data should be written and whether values should be stored at all. The SD card can also be safely unmounted (or mounted) from the system via the web page. The size and the used space of the SD card is also displayed. The data itself is saved as a CSV file in the root directory of the SD card, with a new file being created each day. In order for the data to be associated with the correct time of day, the time must be set under the menu option *Time settings* (**Figure 2**).

A few more settings are necessary to use the cloud services. For SenseBox, and thereby openSenseMap, a *SenseBox ID* and a separate key for each sensor is required (**Figure 6**). These can easily be taken from the openSenseMap web interface. A virtual channel can then be assigned to each of these sensor keys. A total of up to 16 values can be transferred to openSenseMap.
The settings for ThingSpeak (**Figure 7**) are similar, requiring the virtual channels be assigned to the associated fields, while the

SHOPPING LIST

- **Main Board bare PCB**
  www.elektor.com/191148-1
- **Connector board bare PCB**
  www.elektor.com/191148-2
- **Dust/particulate bare PCB**
  www.elektor.com/191148-3
- **2×16-character LCD with I2C coms**
  www.elektor.com/2x16-character-lcd-blue-white-120061-77
- **Mini SD card module**
  www.elektor.com/19251
- **BME280 BoB, I²C-Version** (160109-91)
  www.elektor.com/bme280-mouser-intel-i2c-version-160109-91
- **ESP32-PICO-Kit V4**
  www.elektor.com/esp32-pico-kit-v4
- **Weather station WH-SP-WS02** (Kit with wind speed/direction sensor, Rainfall gauge, Temp/Hygro sensor, installation bracket)
  www.elektor.com/professional-outdoor-weather-station-wh-sp-ws02

WEB LINKS

[1] **Open-Network Weather station Mk.2 (Part 1), Elektor May & June 2020**: http://www.elektormagazine.com/191148-01
[2] **Elektor labs weather station web page for the project**: https://www.elektormagazine.com/labs/remake-elektor-weather-station
[3] **Project page for this article**: http://www.elektormagazine.com/191148-B-02
[4] **Starting out in Node-RED**: https://www.elektormagazine.com/articles/starting-with-nodered
[5] **Practical ESP32 multitasking**: https://www.elektormagazine.com/190182-01

API Key is only required for writing.
MQTT requires significantly fewer settings. However, more information is needed for the user to process the data (**Figure 8**). In addition to the intervals at which the data is sent (and a decision on whether data is to be sent), there is also the item *ioBroker compatibility*. If the data is processed using Node-RED there is no need to use this mode. In such cases, the data is transported via MQTT so that the broker can extract the data from it. In the following example a generic transport via MQTT is assumed with a system that can process a JSON string, such as Node-RED [4].

```
// the JSON produced by the
   Station looks like this:

{
 Data:[
   {channel:0, value:0.0
   ,Name:"WINDSPEED(10S).0.Speed.
   INTERNAL"},
    …
   {channel:63, value:0.0
   ,Name:"WINDSPEED(6d0S).0.Speed.
   INTERNAL"}
      ]
}
```

Only channels that are assigned a sensor value are transferred within the `Data` array. Channels that have no assignment do not appear.

Averaging of measurements

When determining the three measured values of rainfall, wind direction, and wind speed, average values are calculated. For wind direction and wind speed, an average value is calculated over a 10 second, 60 second, and one hour period. Rainfall is calculated over a 30 minute, 60 minute, 720 minute, and one day (1440 minutes) period. The values shown are rolling averages based on the time elapsed. This function currently stores these values in the drivers, but they could be relocated later by the kernel itself to a 'recorder'. This could accept the data as a Connector and, in turn, provide it as virtual channels. This principle could also be used to implement other mathematical functions.

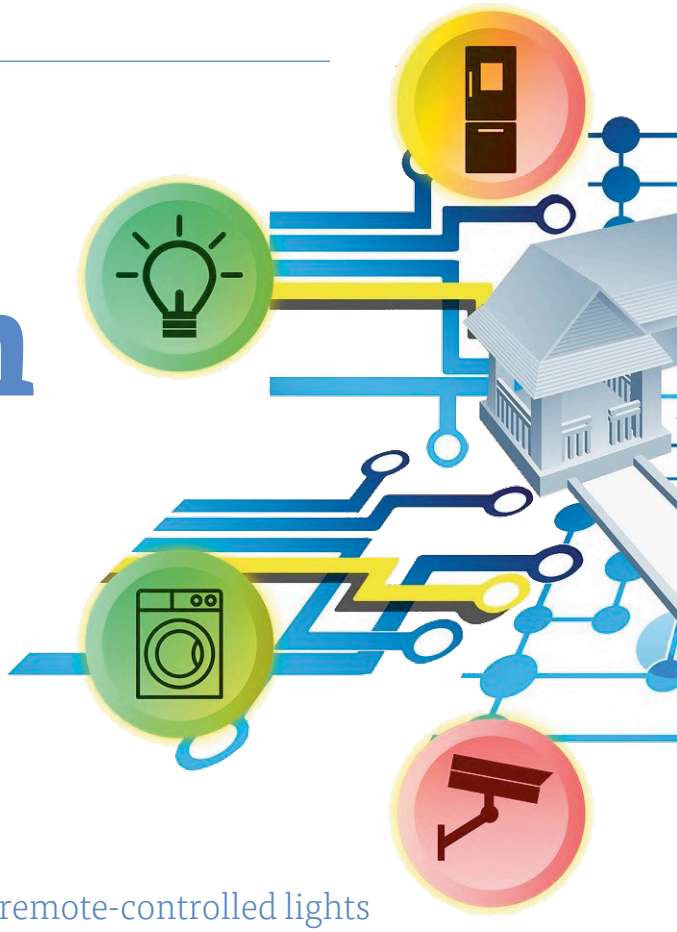
Room for expansion

Even though the weather station is now complete and ready for use, there will surely be one or two requests to expand its functionality. There are already a few ideas bouncing around in the lab, such as adding support for one-wire sensors. A file browser for the web interface would also be interesting, allowing the deletion of files or the download of data directly to the SD card. We welcome any other suggestions you may have - please leave them as comments on the Elektor Labs page for this project [2]!

191148-B-02

Home Automation Made Easy

With ESPHome, Home Assistant & MySensors



By **Clemens Valens** (Elektor Labs)

Who has never dreamt of equipping their home(s) with remote-controlled lights or curtains, or window shutters that open and close automatically? Home automation, that's the name of the game!

I had a dream

I for sure had my share of dreaming, but I never really went much further than that because as soon as I began to realize a dream I invariably ran into those practical hurdles that my motivation apparently couldn't overcome.

Building a wireless battery-operated temperature sensor to measure the temperature in the living room isn't terribly complicated, and adding a remote-controlled relay to switch a heater on and off is doable too. But there the complexity only starts. Some sort of controller is needed to allow the creation of basic rules like "*on weekdays, switch the heater on at 7 in the morning; in weekends, not until 9*". And it must allow for manual override in such a way that every authorized occupant of the house can adjust the temperature without first having to learn Python.

Up to this point, we can't even speak of home automation as this was just the description of a slightly fancy, programmable thermostat. Far more is needed to turn a remote-controlled lamp into a full-blown extendable home automation system. Which is why I never got beyond dreaming.

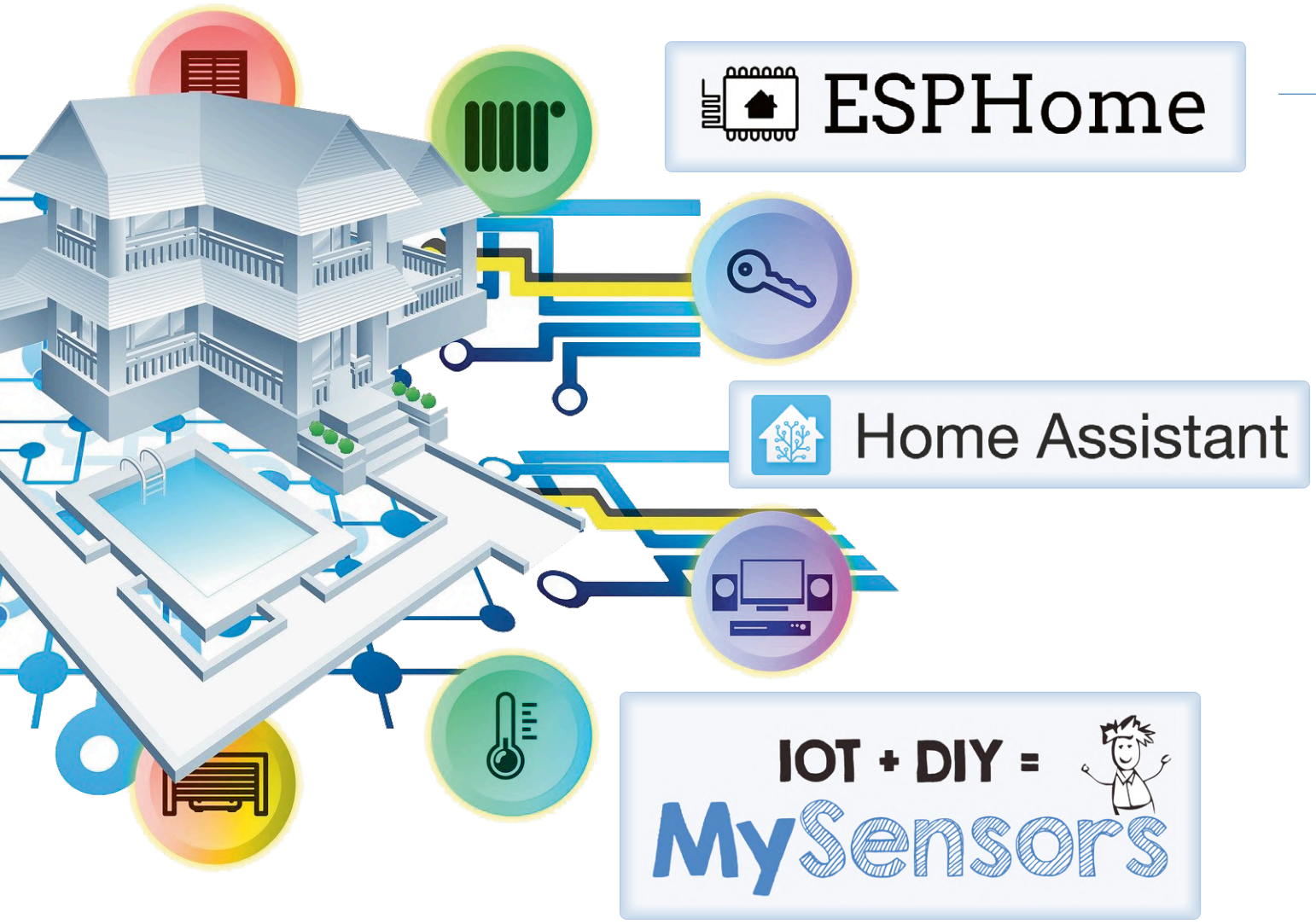
Espurna & ESPHome

Until recently, while searching online for the manual of a Wi-Fi–controlled power plug I bought a few years ago but never used, I came across the 'Espurna' project [1] (even though I ended up not using it, I am mentioning it here as it is an excellent open-source project that deserves to be known). Espurna allows for the easy creation of Wi-Fi based (mainly ESP8266) sensors and controllers that can talk to each other. ESPHome [2] (**Figure 1**) is a project very similar to Espurna.

Figure 1: The rather lengthy ESPHome homepage shows all the ready-made components that can be part of an ESPHome device.

And you thought Arduino was easy?

Espurna and ESPHome both provide easy-peasy ways of programming Espressif's ESP8266 and ESP32 devices. In fact it's so relaxed that they make Arduino look like rocket science. In short — after installing the software, of course — you write a simple text file that



specifies which kind of sensor is connected to which pin(s) of your ESP module and after compiling and flashing the firmware, you end up with a smart device featuring a web interface, over-the-air (OTA) programming, MQTT communication, and what not. Impressive, isn't it? I think it is, but it doesn't stop there.

Take control of your Wi-Fi connected smart plugs

The Internet is afloat with cheap smart plugs and relay boards with on-board Wi-Fi and other Wi-Fi–connected gadgets. Many of these devices are built around the ESP8266 (**Figure 2**). As a common feature, they require an account to a different cloud service somewhere on the Internet, controlled through their own app on your smartphone. This makes them very tedious to use, so much so that these devices often end up in a drawer or, worse, in the bin.

No longer, because Espurna and ESPHome let you reprogram these things with firmware that you can modify at will. No Internet connection required, goodbye broken-English apps and dubious cloud services; just reflash the darn thing and integrate it into *your* home automation system controlled by *you@home*.

From remote control to automation

ESPHome and Espurna also feature automation, allowing users specify rules for switching equipment on and off following event triggers and sensor data. Both enable you to create a pretty fancy home automation system out of commercially available cheap Wi-Fi–capable hardware or stuff built by yourself.

This article could end here if wasn't for the fact that I found doing automation on Espurna and ESPHome overcomplicated. I am willing to blame this on my limited intellectual capabilities, but only partly, as the lack of good documentation for these projects doesn't help either. Even though there seems to be quite a lot of it, it is rather fragmented and not always clear.

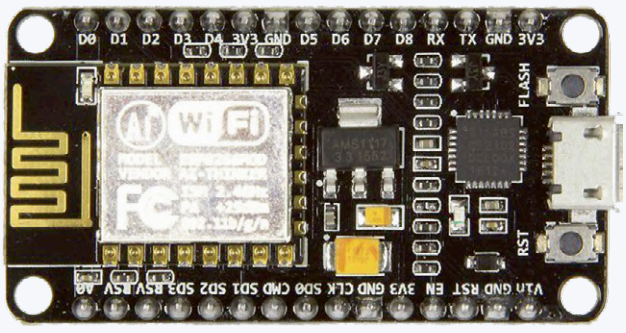


Figure 2: ESPHome runs on ESP8266-based modules like NodeMCU, which is common and cheap. ESP32 modules can be used as well.

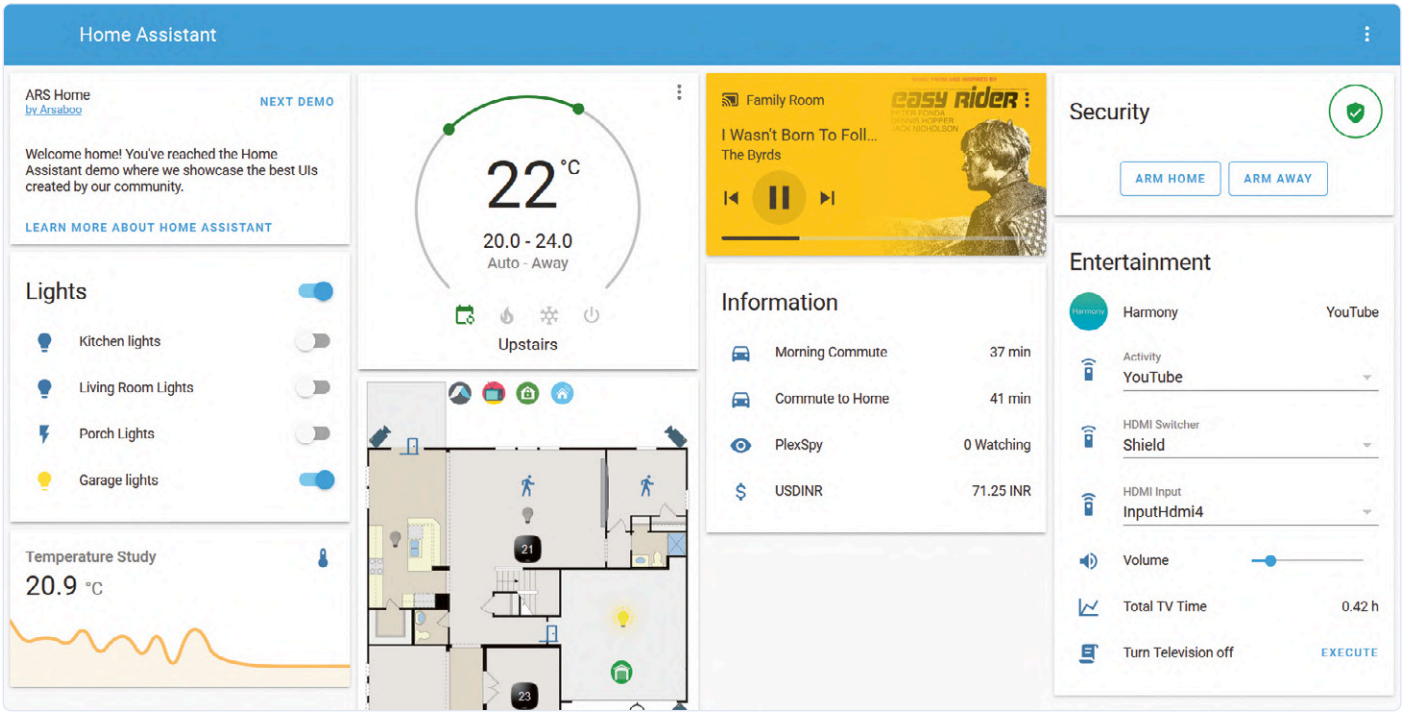


Figure 3: The Home Assistant dashboard is fully customizable and can be as fancy or spartan as you like.

Home Assistant

Both ESPHome and Espurna provide integration with Home Assistant [3], a so-called home automation hub or *controller*. That's a device allowing the user to combine sensors and actuators (from different manufacturers) within a single system. Here 'sensors and actuators' should be regarded in a rather broad sense as they range from thermometers and GPS through Internet services right up to motor control and texting. Home Assistant (also 'HA', 'Hass' or 'Hass.io') is compatible with — to cite a few well-known names — Alexa and OK Google, Ikea's Trådfri, Philips's Hue, Z-Wave and Zigbee Home Automation, and Sonoff by iTead [4]. At the time of writing, HA lists 1574 integrations and goes way beyond the possibilities offered by Espurna and ESPHome.

You use HA to define the rules to govern the appliances in and around your home. "*If the sun goes down in 20 minutes and the smartphone of occupant A is inside the house and it hasn't rained for three weeks, then switch on the third sprinkler from the right.*" This is typical (and even basic) HA automation stuff — assuming the concerned hardware is installed and working as intended. HA also provides a fully customizable graphical user interface (GUI) or dashboard for the system (**Figure 3**). HA is open source, written in Python and it runs on a Raspberry Pi. It also works on other operating systems; it's just that I installed it on a Raspberry Pi 3 because that was so easy. You will have understood by now that I am a sucker for easy.

Integration in Home Assistant

Home Assistant was also the reason why I continued with ESPHome instead of Espurna. ESPHome has a plugin (**Figure 4**) that lets it be integrated in Home Assistant in such a way that ESPHome-based devices are detected automatically by HA. Flash'n'Play, what more do you want? The integration is pushed so far that you don't even need a development computer anymore; sensors and actuators can be (re)programmed and (re)configured from your smartphone while sitting in your comfy chair.

Low-power IoT devices

Wi-Fi is great for quickly connecting devices to a network, but it is a tad power-hungry. It is therefore not the best solution for low-power sensor nodes required to run many years off a single button cell, or living off energy harvesting. Such devices spend most of their time sleeping, and when they wake up, they spit out their data as fast as possible since they don't have the energy for engaging in long handshaking protocols.

A great solution for this type of device is MySensors [5], an open-source home automation and IoT project based on ISM-band radios, notably the nRF24 by Nordic Semiconductor (**Figure 5**) and HopeRF's RFM69. The more recent nRF5 platform as found on the BBC micro:bit can be used too.

The MySensors website is a slightly messy but once you manage to see through the noise, you'll notice that there is pretty good stuff available. MySensors uses mainly (but not only) Arduino as microcontroller platform, and it builds and maintains a tree (or star) network all by itself. Like ESPHome it integrates smoothly (but not as smoothly) with Home Assistant.

The project comes as an Arduino library included in the Arduino IDE's library manager. After installing it, you can build your application on one of the examples. In many cases this only means changing the pin number(s) of the connected peripheral(s).

Getting your hands dirty

After this long introduction you may want to get practical and for this I'd suggest fist setting up Home Assistant on a Raspberry Pi. A good

step-by-step guide is available at [3] under the 'Getting Started' tab. There the use of a Pi 4 is suggested but I have HA running on a Pi 3. A 32 GB or bigger microSD card is recommended. Do realize that the Pi cannot handle exFAT-formatted SDXC cards (i.e. SD cards bigger than 32 GB), so if you opt for a 64 GB microSD card (or even bigger), make sure to (re)format it as FAT32. For a more robust system you may want to use an SSD instead of a (fragile) microSD card.

Multicast DNS

Home Assistant relies on the multicast DNS (mDNS) protocol to find devices and communicate with them, but this protocol isn't particularly well supported by Android and Windows (see inset). Apple devices and Raspberry Pi work fine and I assume computers running some other form of Linux will work too. For this reason, you may want to choose a static IP for HA. My HA system uses DHCP and I have experienced connection problems with the HA app for Android when the HA computer was given a new IP by the DHCP server.

Note that the HA installer quickly launches a web server where you can monitor the installation progress; connecting a display to the Pi isn't useful here. Consult your network's router to find the IP address of this server.

Gain access to the configuration file

The second installation step is to configure Home Assistant. It has a wizard for this, so I won't go into details here. More important is the installation of some add-ons after configuring HA. You do this from the 'Supervisor' menu under the 'Add-on Store' tab. For some reason the default HA setup does not allow you to edit its main configuration file (*configuration.yaml*) yet you will find yourself having to do it regularly, especially when you are experimenting with the system. I therefore installed the 'File Editor' add-on (formerly known as 'Configurator') and also 'Samba share'. The first lets you edit low-level files directly in HA, the second exposes some HA folders on the network, allowing you to e.g. edit files with your favourite text editor. For security reasons you may want to install the 'Terminal & SSH' add-on instead.

Samba also allows you to use HA as a file server if you create a 'www' folder in the 'config' folder.
Installing add-ons is straightforward, just click on the add-on's card to open it and then click install. An opened card contains instructions on how to configure the add-on.

Installing the ESPHome add-on

This brings us to ESPHome, because we should also install the ESPHome add-on. To get this done, first add the repository's URL to the store. Then locate the box saying 'Add new repository by URL' and paste the following URL into it:

```
https://github.com/esphome/hassio
```

Now you can install the ESPHome add-on. There are three versions: plain, beta and dev; we will use the plain vanilla version. The only configuration I did for this add-on was enabling the 'Start on boot', 'Auto update' and 'Show in sidebar' options. The latter is practical as it makes accessing the add-on much easier.

Integrating MySensors

At the time of writing, there was no Home Assistant add-on for MySensors. Activating this integration in HA is therefore done by adding a few lines to the *HA configuration.yaml* file (see inset):

```
mysensors:
   gateways:
   - device: '192.168.1.100'
     persistence_file: 'mysensors/wifi_gateway.json'
     tcp_port: 5003
     optimistic: false
     persistence: true
     retain: true
     version: '2.0'
```

As indicated, you have to choose a static IP for the MySensors gateway

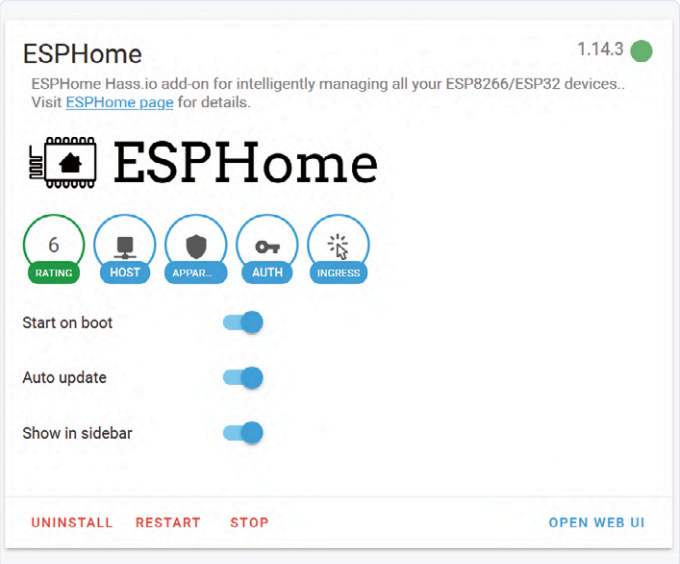


Figure 4: I like to have the 'Show in sidebar' option activated for the ESPHome add-on for Home Assistant.

Figure 5: nRF24 modules come in different shapes. The 8-pin type appears to be the most common; 10-pin types are identical except for the pinout.

ENABLING **MDNS** ON WINDOWS

In the case of a DHCP network configuration (i.e. when the network attributes IP addresses to connected devices) Windows users may experience difficulties connecting to HA because they do not have the multicast DNS (mDNS) protocol running. Having mDNS running enables you to connect to HA by using the address http://hassio.local:8123 (at [3] other URLs are given, but they didn't work for me). Online I found a way of fixing this:

1. Open the Registry Editor (Windows key + 'R', then type 'regedit') and navigate to
   `Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows NT\DNSClient`
   Right-click on it and choose 'New' to create a new 32-bit DWORD named 'EnableMulticast' with a value of 0 (zero).
2. From the Apple website, download but do not install iTunes. This requires a bit of searching. The last time I checked, the way to do this was by choosing 'iTunes for Windows' and then not to get it from the Microsoft Store, but scrolling down instead until you see 'Looking for other versions?'. That link will take you to the file you want. After the download has completed, change the extension 'EXE' to 'ZIP' and extract from it the file 'bonjour.msi' (or 'bonjour64.msi', depending on the iTunes version you downloaded).
3. Install 'bonjour' and restart your computer.

https://www.apple.com/itunes/

A WORD ABOUT **YAML** INDENTATION

Both Home Assistant and ESPHome use YAML (Yet Another Markup Language) for their configuration files. Like Python and ancient assembly languages, this format uses indentation to structure information (a horrible way of doing things, if you ask me). Indentation must be the same for all the lines at the same level, but every level may have different indentation. For the YAML snippets in this article every level of indentation is two (2) spaces.

you are going to build (yes, you are, see below), and you have to choose a name and location for the JSON file where HA can store MySensors network information.

Important: Every time the HA configuration.yaml file is modified, the system needs a restart for the changes to take effect. This can be done from the Supervisor's 'System' tab.
Now that you're editing the configuration file, you might as well add the following (saving a restart):

```
binary_sensor:
  - platform: workday
    country: [country code]
```

These lines enable writing automation rules that only fire on workdays or weekends, things like that. Replace `[country code]` by the code of the country the system is working in. Refer to the help page of the 'Workday Binary Sensor' to find this code (and other useful information).

My first ESPHome device

If you followed the instructions above, then you are now set to program ESP8266- and ESP32-based devices. Because a new (virgin) device is not yet compatible with ESPHome, initially it has to be programmed over the serial port. Depending on the device, you may have to install a USB-to-serial-port driver on the HA computer to make this work. The NodeMCU boards with a Silabs (Silicon Laboratories) CP2102 USB chip that I used worked straight out of the box.
After connecting the device to the computer running HA, open the ESPHome Dashboard either from the sidebar (if you enabled this option) or by clicking 'Open Web UI' from the add-on card in the Supervisor Dashboard view. Check that the serial port is available from the drop-down list in the upper-right corner that defaults to 'OTA (Over-The-Air)'. If it doesn't, restart the ESPHome add-on (by going back to the ESPHome card on the Supervisor Dashboard), that should do the trick.

The next step is to click the pinkish '+' button on the ESPHome Dashboard. This will open a wizard to guide you through the first part. Note that I have never specified an access password for any device, but maybe I am being irresponsible.
For the upload port, select the serial port to which the device is connected.

Editing the YAML file

The wizard is now done and a card should have been created for your device. At this point clicking its 'Edit' button never works (for me), but after reloading the page it does. Click it and check the credentials for your Wi-Fi network and make sure you see the lines 'api:' and 'ota:'. If you upload this configuration to your device and reboot it, it will be detected by Home Assistant. It can now also be disconnected from the computer because over-the-air programming has been enabled. The device will not do anything, though, because you did not configure its peripherals. Therefore, read on before uploading a configuration.
If you use a NodeMCU module, you can kickstart it by adding the lines below to the end (below 'ota:' for clearness sake, the position in the file doesn't matter) of the configuration file before uploading it. It will give HA access to the on-board LED and the 'Flash' pushbutton.

```
output:
  - platform: gpio
    id: "led"
    pin:
```

```
      number: GPIO16
      inverted: True

light:
  - platform: binary
    name: "LED"
    output: "led"

binary_sensor:
  - platform: gpio
    name: "Flash pushbutton"
    pin:
      number: GPIO0
      inverted: True
```

Note that in this snippet every level of indentation is two (2) spaces, meaning that the lines 'number: GPIOx' and 'inverted: True' start with six spaces (as they have three levels of indentation, see inset).
This config can also made to work with an ESP-01 module if you change 'GPIO16' to 'GPIO3' and connect an LED in series with a 470 Ω (or so) resistor between GPIO3 (RXD) and ground. Also connect a pushbutton between GPIO0 and GND and a 10 kΩ (or so) pull-up resistor between GPIO0 and 3V3. Do not press this button when booting the device.
Upload this configuration to the ESPHome device and wait for it to restart. Home Assistant should see it — the ESPHome dashboard should show it as 'Online' — and, if you let it, create a control for a light and an indicator for the pushbutton. If you uploaded the empty configuration first, then you may have to go into HA's 'Configuration' menu to look it up in the 'Devices' list.

You can now add cards for the controls to HA's 'Overview' (click the three dots and then 'Configure UI') and create automations in the 'Automation Editor' on the Configuration menu ('Automations'). I leave this to you as it is quite self-explanatory and a good excuse to browse around in Home Assistant. Also, MySensors is waiting for us.

Build a MySensors Wi-Fi gateway

A gateway is required to build a MySensors network and you also need it to connect to other (Wi-Fi) networks. For this I used an ESP8266-based NodeMCU because it exposes an SPI Port which we are going to need. Using an ESP32 module is another option. Actually, there are many options, but in this article we are going for Wi-Fi.
Connect the SPI port to an nRF24L01+ module (**Figure 6**). It is recommended to also add an electrolytic capacitor (4.7 to 47 µF) together with a ceramic one (100 nF or so) between the VCC and GND pins of the nRF24 module. That's all the 'electronics' you'll need.
On the software side a computer with the Arduino IDE installed is required. Add to the IDE the ESP8266 (or ESP32) core for Arduino — all the details are at [6] — and the MySensors library ('Sketch' ➞ 'Include Library' ➞ 'Manage Libraries…'). Load the example 'GatewayESP8266' ('File' ➞ 'Examples' ➞ 'MySensors') and enter the correct SSID, password, and the static IP that you specified earlier in HA's configuration.yaml file. Upload the sketch to your device, and your gateway is ready.
Please keep in mind that a gateway (and repeater nodes, not treated in this article) must always be powered and may never sleep. Sensor nodes can do whatever they want.

My First MySensors Node

Creating a MySensors node is not unlike building a gateway, except that the ESP module is replaced by an Arduino-compatible board with

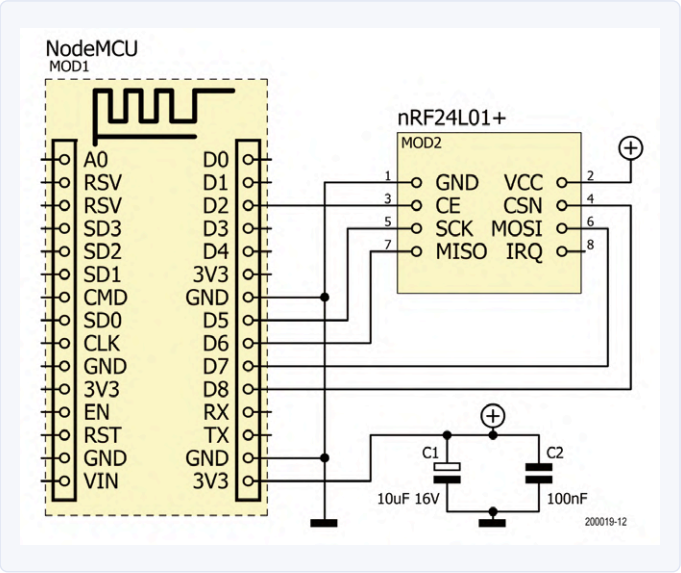


Figure 6: A MySensors Wi-Fi gateway can be built using a NodeMCU module.

sensors and actuators added to it. Connect an nRF24L01+ module to the board's SPI Port, and add the capacitors mentioned above (**Figure 7**).
Load an example sketch from the MySensors example library, preferably one that resembles what you are trying to achieve. Note that there are more examples on the MySensors website. The sketch requires adapting to fit your peripheral(s) pin number(s), but basically that is all there is to do. Upload the sketch to your device and let it boot; it will join the MySensors network automatically (if the gateway is on). Really.

Beware of the protocol version

At this point I ran into the problem that my device, a remote relay, did not show up in Home Assistant. Eventually I discovered that this had something to do with the version of the MySensors API being used. By inspecting the persistence JSON file (see section 'Integrating MySensors'), I noticed that the gateway was listed as using API or protocol version 2.3.2. The MySensors page at [7] has an example intended for use with API versions 2.x. When I tried that one, my node appeared in the 'Entities' list on HA's 'Configuration' menu and I was able to create UI cards for it.

According to [7], to make a V2.x-based node work in HA, it must send an initial value from the function `loop()`. The example sketches do not do this. However, delving deeper I surmised that an *actuator* node must *request* (and maybe also send) an initial value from HA in order to be recognized. A sensor node is reported as soon as it starts sending data. The request doesn't have to be made from `loop()`, but simply somewhere during the boot process.

Special functions 'before()' and 'presentation()'

To distinguish V2.x-sketches from older types, look for the function `presentation()`. If the sketch has one, it is V2 or higher. However, the other way around is not true as `presentation()` may be left out. The `present` instructions usually found in this function may not be omitted and must be moved to another function, for instance to `setup()`.

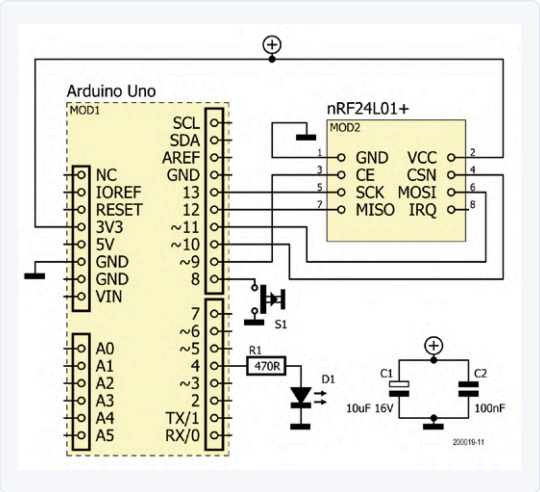


Figure 7: The schematic of a MySensors node with a pushbutton for a binary sensor and an LED for an actuator.

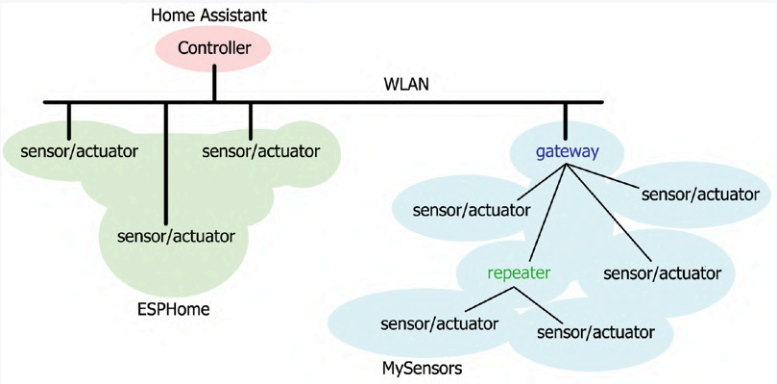


Figure 8: When you combine all the elements described in this article you will end up with a home automation system like this.

MySensors sketches can have a function `before()`. Both `before()` and `presentation()` are called before `setup()`, and in this order, i.e. `before()`, `presentation()`, `setup()`, and finally `loop()`.

Adapting existing code to your hardware is now quite easy. Many examples for all sorts of popular sensors are available, either in the Arduino library or online, so look around before diving in head-first.

OK that's it, I hope you found it interesting

Here endeth the introduction to easy home automation (**Figure 8**). Please keep in mind that it is not complete and that there may be better or easier ways of doing things. My understanding of the components involved is still evolving.
Furthermore, there are many other home automation projects that can do the same or similar things as the ones mentioned in this article. Some may be better, others may look nicer; they all have their strong and weak points.
Because there are so many, it's hard to choose which one(s) to use. In this article I have presented a few that I have tried and kept to build my home automation system on. I still use them, and I am still amazed by the endless possibilities they offer. Stay tuned, because I do not plan to stop here.

At Elektor Labs you can find example configurations for ESPHome [8] and MySensors devices [9]. There are also some videos.

200019-01

WEB LINKS

[1] **Espurna**: https://github.com/xoseperez/espurna
[2] **ESPHome**: https://esphome.io/
[3] **Home Assistant**: https://www.home-assistant.io/
[4] **Sonoff**: https://www.itead.cc/
[5] **MySensors**: www.mysensors.org/
[6] **ESP8266 core for Arduino**: https://github.com/esp8266/Arduino
[7] **MySensors in Home Assistant**: www.home-assistant.io/integrations/mysensors
[8] **ESPHome at Elektor Labs**: www.elektormagazine.com/labs/how-to-home-assistant-esphome
[9] **MySensors at Elektor Labs**: www.elektormagazine.com/labs/mysensors-home-assistant-howto

The Storage CRT

Peculiar Parts, the series

by **Neil Gruending** (Canada)

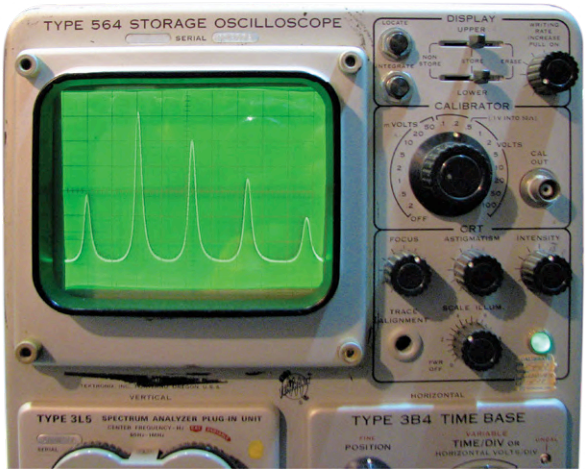


Image: TekWiki

Traditional cathode-ray tube (CRT) oscilloscopes are a great tool for observing repetitive signals, but slow sweeps and sporadic signals are a challenge. Modern digital oscilloscopes solve these problems by using a digital sample memory to store the waveforms, but early analog oscilloscopes needed a different solution. One popular approach was to use the CRT itself as the storage memory. These were known as storage or bistable CRTs.

A storage CRT, such as that shown in **Figure 1**, takes advantage of the fact that the velocity of electrons flowing between two surfaces is controlled by the voltage difference between them. It integrates two electron guns: a writing gun and a flood gun. The writing gun is like a regular CRT gun, writing the waveform to the CRT phosphor. The difference to a conventional oscilloscope is that its drive voltage is high enough to create a positive charge in the phosphor. This positive charge is created when the electrons are moving fast enough that, when they strike the phosphor, more electrons are released than are hitting them.

Once the waveform has been written it is displayed using the flood guns. These hit the phosphor with a stream of low-energy electrons over the entire surface. The electrons hit the unwritten surface too slowly to release many electrons, so these areas are slightly negatively charged and the current flow stops. But the positively charged areas, where the waveform was written, attract electrons and accelerate them. When they hit the phosphor surface these areas release the same number of electrons as hit them. This maintains the positive charge and creates a current flow to illuminate the trace.

However, there is a problem with this method because the saved waveform would, over time, spread and bleed like ink on wet paper. The Hughes Memoscope 104 used several mesh grids in the target to limit the effect, but it lacked contrast and sharpness. It was also very expensive to manufacture. Then, in 1959, Bob Anderson started to look for a lower-cost alternative for Tektronix. Knowing that the phosphor needed to have a discontinuous or broken target surface, he initially tried using dot patterns in the phosphor. Unfortunately, it was difficult to produce consistently, but this approach led him to develop a porous layer of scattered phosphor particles on the target as the storage medium (**Figure 2**). This new tube was first used in the Tek 564 oscilloscope (1963). The design was quickly followed by the 549 oscilloscope that featured a split-screen that could store two different waveforms. Tektronix also used the technology for large screen displays like the 4000 series' computer terminals. It's hard to find these gems nowadays since digital oscilloscopes and TFT displays have taken over. But if you can find one, keep hold of it with the knowledge of all the hidden engineering inside.

190383-E-01

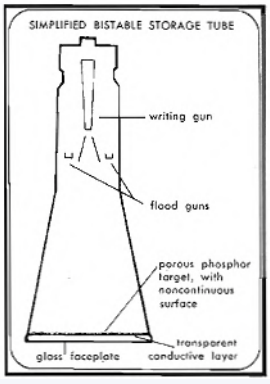


Figure 1. Simplified diagram of a storage CRT [1].

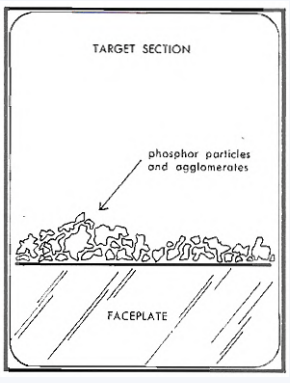


Figure 2. Porous phosphor target layer [1].

WEB LINK

[1] **Vintage Tek - The Storage Story**: https://vintagetek.org/wp-content/uploads/2011/10/The-Storage-Story3.pdf

Artificial Intelligence for Beginners (3)

A stand-alone neural network

By **Walter Trojan** (Germany)

In the previous two instalments of this series we looked at the Maixduino development board. We also demostrated how to use MicroPython to run a pre-built neural network capable of detecting faces in an image. In this third and final instalment we will develop a stand-alone neural network. It also makes use of the on-board ESP32 which, in this application, functions as a co-processor to support with communication tasks.

So far I have shown how to run well-known AI examples using neural networks (NN) that were already trained on the Maixduino. If you want to implement your own NNs, you still have to carry out the development and training on a PC or a cloud service, transferring the resultant trained NN to the Maixduino. This process is not necessarily a disadvantage because training usually involves large quantity of data that benefits from computers with high processing power. This trained NN can then be ported to a low-power mobile platform implementing, for example, a robotics controller.

Here I describe the path from the development of an NN to its execution on the Maixduino. We start by installing the development environment on a Linux PC. The setup shown in **Figure 1** shows the tools belonging to the application development platform.

AI building blocks

Keras is an open-source AI framework that provides the API to the development system. It is a user-friendly, modular, and extensible library written in Python providing a 'LEGO bricks for AI' approach. It can run on top of the very popular TensorFlow framework or on alternative systems such as Theano. If the development PC uses an Nvidia graphics card, you can make use of its many GPU cores with the CUDA computing platform and programming model from Nvidia. Alternatively, you can use BLAS routines running on the PC processor.

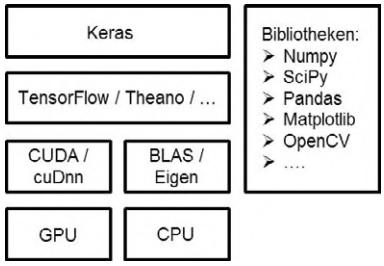


Figure 1: Structure of a popular AI development environment.

Here, again, is a brief description of the main tools used:
Keras: An AI framework that runs on top of TensorFlow or other frameworks.
TensorFlow: The powerful 'go-to' AI framework developed and used by Google.
CUDA / cuDnn: Library that supports Nvidia graphics processors.
BLAS / Eigen: Programming library including elementary operations used in linear algebra, such as vector and matrix multiplication.

Then there are additional library resources that can be integrated as required:
NumPy: Provides highly efficient mathematics functions, especially for matrix operations.
SciPy: Library based on *Numpy* providing additional functions, e.g. differential equations.
Pandas: Useful for manipulating numerical tables.
Matplotlib: A plotting library to draw diagrams and graphs.
OpenCV: Provides powerful image processing support.
HDF5: An open source file format that supports large, complex, heterogeneous data.
Graphviz: Used to create visual representations of structured data.
pydot-ng: Helps with use of Graphviz capability.
Jupyter: An IDE for Python. Program sections are structured as cells which can execute individually. Preferred IDE for professional developers.

Other AI frameworks, such as Theano, Torch, Caffe, Pytorch and Yolo, are also available along with additional libraries. These lists provide just a tiny fraction of the material available to developers working in the Linux/Python world. If you have a particular application in mind, it's worth spending some time exploring relevant resources already

available on the web. The environment shown in **Figure 1** can be set up on a PC running Linux using the following terminal commands:

› Update packages:
```
sudo apt-get update
sudo apt-get upgrade
```

› Python and pip:
```
sudo apt-get install python3-pip python3-dev
pip install --upgrade pip
```

› OpenBlas:
```
sudo apt-get install build-essential cmake git unzip
   pkg-config libopenblas-dev liblapack-dev
```

› SciPy, Numpy:
```
sudo apt-get install python-numpy python-scipy
   python-yaml
```

› matplotlib:
```
sudo pip3 install matplotlib
```

› HDF5:
```
sudo apt-get install libhdf5-serial-dev python-h5py
```

› Graphviz, pydot-ng:
```
sudo apt-get install graphviz
sudo pip3 install pydot-ng
```

› OpenCV:
```
sudo apt-get install python3-opencv
# installs Version 3, Version 4 more complicated to install
```

› TensorFlow 2.0:
```
pip3 install tensorflow==2.0.0b1
```

› Keras:
```
sudo pip3 install keras
```

› Pandas:
```
sudo apt-get install python3-pandas
```

› Thonny:
```
sudo apt-get install thonny
```

› Jupyter:
```
pip3 install jupyter
```

› Invocation:
`jupyter notebook` (terminal command)

The essential building blocks are now in place. If additional project-specific components are required, they can be installed using `pip` or `apt-get`. Thonny is the Python editor used here. Its features are fairly basic but it provides sufficient functionality for our purposes. Professionals would generally use alternatives such as Jupyter that allows program structuring in cells that can also be exported individually. Once the above steps are complete, we are ready to develop an NN from scratch and transfer it to the Maixduino. As our first hand-built

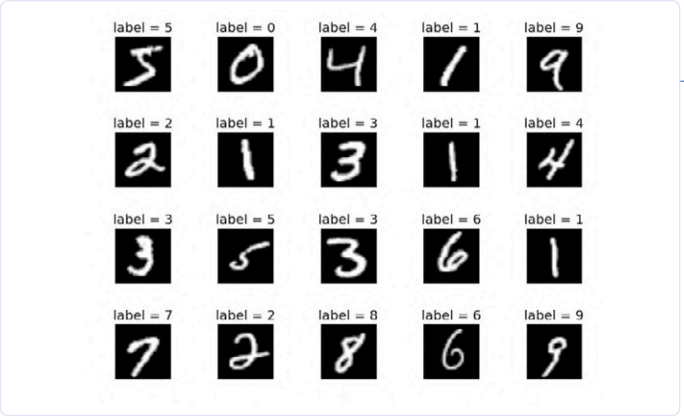


Figure 2: An extract from the MNIST database.

example of a useful NN, I decided to create an application that recognises handwritten numbers. It is trained using the MNIST database (the Modified National Institute of Standards and Technology database). The database contains 60,000 sample images for training purposes and a further 10,000 images for testing. Furthermore, it is available for free. All of its 28 × 28 pixel images are labelled with the numerical values they represent (**Figure 2**).

Step by step to NN

The NN created using Keras/TensorFlow should be able to correctly classify handwritten numbers that are not yet known to it. The development phases of an NN typically follow these steps:

› **Data preparation**: Convert training and test data into a suitable format.
› **Definition**: Define the NN structure with the type and number of layers.
› **Compilation**: Conversion of Keras instructions into TensorFlow commands.
› **Training**: Iteratively process the training data to optimise the NN weights.
› **Test**: Check prediction quality using the test data.
› **Recognition**: Submit new (unknown) handwritten number images for classification.

Data Preparation

The MNIST files are already available in the Keras library and are imported in the appropriate format.

Definition

The NN structure (`model`) that follows, developed by professionals, already offers an excellent detection rate of over 99%.

```
model = Sequential()
# Use sequential model (alternative is a functional
# model)
model.add(Conv2D(32, kernel_size=(3, 3),
# First two-dimensional convolutional layer with
    # 32 outputs
                activation='relu',
      Define activation function
                input_shape=input_shape))
     # Input format 28 × 28 pixels
model.add(Conv2D(64, (3, 3), activation='relu'))
```

THE ESP32 - OUR LITTLE HELPER

Although the K210 chip is a powerful processor for AI applications, it does not have any built-in ability to handle analogue inputs or communications via Wi-Fi or Bluetooth. This is where the ESP32 comes in. In addition to these functions, it also incorporates a dual-core processor running at a clock frequency of 240 MHz. The interface between the K210 and ESP32 takes place via a fast SPI channel, and six analogue inputs are made available on the connectors following the familiar Arduino standard. A serial connection to the ESP32 can be established using the *ttyUSB1* port and the ESP32 is programmed with the MaixPy-IDE via the K210.

Here is an example that shows how to measure analogueue input signals:

```
import network
   # import necessary modules
import utime
from Maix import GPIO
from fpioa_manager import *

#iomap at MaixDuino
   # register GPIOs for ESP32 interface
fm.register(25,fm.fpioa.GPIOHS10)    # cs
fm.register(8,fm.fpioa.GPIOHS11)     # rst
fm.register(9,fm.fpioa.GPIOHS12)     # rdy
fm.register(28,fm.fpioa.GPIOHS13)    # mosi
fm.register(26,fm.fpioa.GPIOHS14)    # miso
fm.register(27,fm.fpioa.GPIOHS15)    # sclk

# definition of network interface
nic = network.ESP32_SPI(cs=fm.fpioa.GPIOHS10,rst=fm.fpioa.GPIOHS11,rdy=fm.fpioa.GPIOHS12,
mosi=fm.fpioa.GPIOHS13,miso=fm.fpioa.GPIOHS14,sclk=fm.fpioa.GPIOHS15)

adc = nic.adc( (0,1,2) )
    # get ADC0 ADC1 ADC2
print('ADC 0,1,2')
    # print results of ADC 0-2
print(adc)
print()
print('ADC 0,1,2,3,4,5')

while True:
    try:
        adc = nic.adc()
    # get ADC0-5
    except Exception as e:
        print(e)
           # in case of error print reason
        continue
    for v in adc:
        print("%04d" %(v), end=" ")
           # print results of ADC 0-5 formatted
    print()
    utime.sleep_ms(1000)
           # pause 1000 ms
```

```
    # Second convolutional layer with 64 outputs and
    # relu activation function
model.add(MaxPooling2D(pool_size=(2, 2)))
    # Maximum pooling layer
model.add(Dropout(0.25))
    # Improve robustness
model.add(Flatten())
    # Conversion from two to one dimension
model.add(Dense(128, activation='relu'))
    # Third NN layer with 128 outputs and relu
    # activation function
model.add(Dropout(0.5))
    # Improve robustness
model.add(Dense(num_classes, activation='softmax'))
    # Forth NN layer with 10 outputs and softmax
    # activation function
```

At the start of the program, the required Python modules are imported. The GPIOs for communication with the ESP32 are then registered and assigned to the `nic` object. The `nic.adc` function has a tuple of the desired channels, in this case ADC0 to ADC2, to measure analogue input voltage levels. After these results have been displayed, all six available inputs are queried in the `while` loop without needing to specify the channel numbers. If errors occur during this process, an error code is output in the `except` statement.

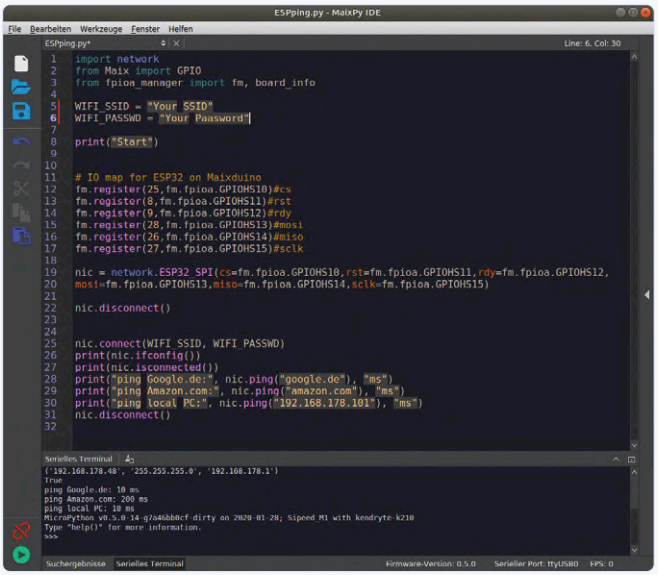
To test the analogue input measurement function I connected the input of channel ADC0 with a potentiometer to provide a variable DC input signal. The voltage range of this input level should be limited to between 0 V and 1.0 V. The values shown in the terminal confirmed correct functionality:

```
ADC 0,1,2
(0, 1786, 73)

ADC 0,1,2,3,4,5
0000 1469 0082 0521 0120 0001
    # Poti auf 0 V
0000 1813 0160 0606 0291 0000
2622 2135 0210 0630 0323 0000
    # Poti Mittelstellung
4095 2421 0259 0575 0261 0000
4095 2472 0274 0615 0315 0000
    # Poti auf 1.0 V
```

The remaing ADC channels have no connections at their inputs, so they display random values in columns 1 to 5.

In this second example, communication via the Internet is confirmed by pinging the well-known Internet portals Google and Amazon, as well as a local computer (see screenshot).

The import of modules, as well as the registration of GPIO ports and the definition of the `nic` objects, are carried out as in the first example. To access the Internet you will need an SSID and Wi-Fi password to connect to your router. Once the connection is established, `nic.ifconfig()` can be used to find the IP address of the ESP32 and the connection status can be displayed using `nic.isconnected()`. The `nic.ping()` command is used to define the address of the target to ping. In addition to website URLs, computers connected to the local network can also be pinged by using their IP address.



Further use cases include transmitting AI analysis to a web server, recognition of objects found on the Internet, and much more. Check out some of the examples in [5] for further inspiration and experimentation. Please note that the Maixduino is still a relatively new platform and is at an early phase of its development. In addition to the firmware for the K210, an update to support the connection to the integrated ESP32 is occasionally required. This can be undertaken using the following command sequence after downloading the binary file from link [4]:

```
pip install esptool
esptool.py --chip esp32 --port /dev/ttyUSB1 erase_flash
esptool.py --chip esp32 --port /dev/ttyUSB1 --baud 1500000 write_flash -z 0x0000
   maixduino_esp32_firmware_v1.4.0.bin
```

This NN makes use of four layers. Two are convolutional layers, while the other two are classic layers (`Dense`) with a manageable number of nodes (32/64/128/10). The activation in the final layer using the `softmax` function limits the sum of the output values to a probability of 1. In the first two layers, blocks of 3 × 3 pixels are used to filter out details such as lines, arcs and dots.
Filters between the layers implement further functions. `MaxPooling` compacts the output from a layer and transfers the average values into, in this case, a 2 × 2 field. `Dropout` randomly drops out nodes from a layer during training that results in more weight be given to neighbouring nodes. This makes the NN more robust. `Flatten` converts a multi-dimensional CNN output into one dimension.

Compilation

In this phase, Keras commands are converted into TensorFlow instructions.

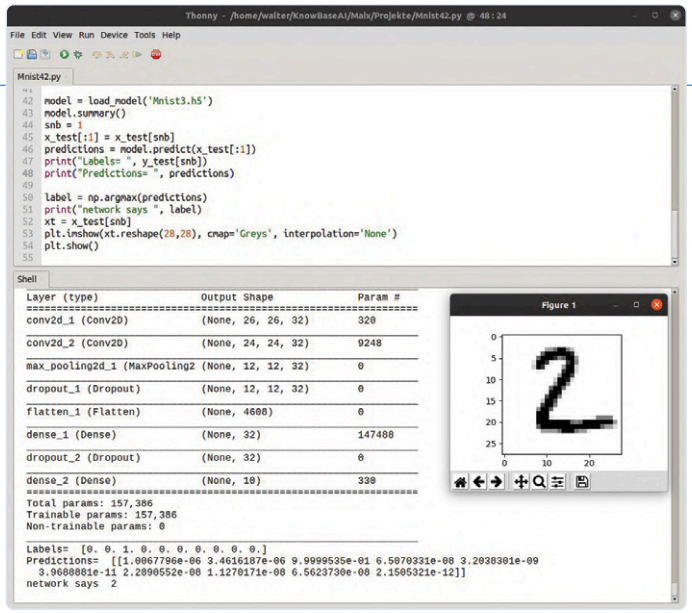


Figure 3: Recognition of the number 2.

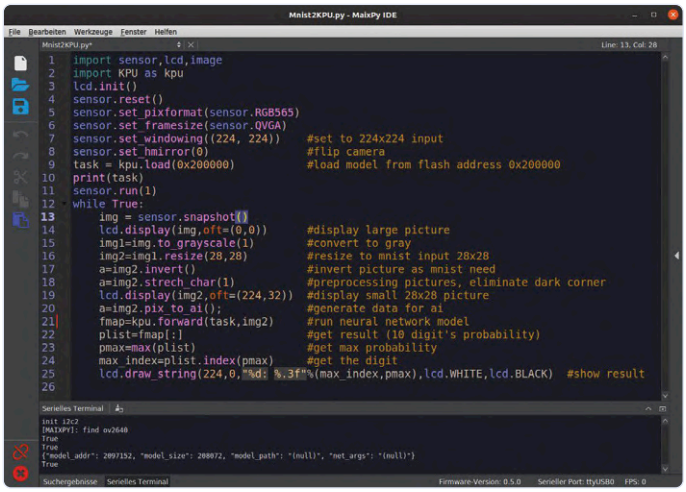


Figure 4: Number recognition with MicroPython.

```
model.compile(loss=keras.losses.
   categorical_crossentropy,
     # Categorise loss function
              optimizer=keras.optimizers.Adadelta(),
     # Select the optimiser
              metrics=['accuracy'])
     # Define accuracy
```

Training

Here the NN is trained using 12 cycles (epochs) on a dataset of 60,000 images of numbers.

```
model.fit(x_train, y_train,
# x_train contains images, y_train contains labels
          batch_size=batch_size,
# Training performed in batches, e.g. in batches of 16
          epochs=epochs,
# Number of epochs to cycle through entire dataset
          verbose=1,
     # Display progress of training
          validation_data=(x_test, y_test))
     # Validate the model with the test data
```

Test

Finally, the detection rate is determined using the 10,000 test images.

```
score = model.evaluate(x_test, y_test, verbose=0)
model.save("Mnist3.h5")
```

By saving the model's structure and trained weights in *h5* format, it can be reused for recognition even on another PC or system.

Recognition

The ultimate goal is to perform number recognition on the Maixduino board once the prepared NN model has been transferred to it. However, it can first be tested on a Linux PC using an editor such as Thonny (**Figure 3**).

To detect new handwritten numbers the trained NN is first loaded, whereupon the model outputs its response. You may have noticed that the number of nodes has been reduced in some layers. This was necessary so that the NN fits into the smaller main memory (5.9 MB maximum for this model). The second element in the test data is, coincidentally, a '2', which is confirmed by the label for this data record. The NN concurs with this labelling and delivers a probability of 0.999 for this index. It also calculates a very low probability value that the image could be of one of the other numbers it has been trained to detect. To provide a meaningful output, the function `argmax` in *Numpy* is used to return the indices of the maximum value from the array. The input image is drawn with the help of the *matplotlib* library. With that, the model is now trained and tested on the Linux PC. So, all that's left to do is transfer the trained NN to the Maixduino.

The steps for the transfer process are:

- Creation of a model trained and tested using Keras, the result of which is an AI model in *h5* format (as just described).
- Conversion of the *h5* model into *tflite* format using TensorFlowLite.
- Use tha *nncase* compiler to convert the *tflite* model into the *kmodel* format for the KPU.
- Use Kflash to flash this model into the Maixduino memory at address 0x200000.
- Create a Python script using MaixPy-IDE and download it to the Maixduino where it can be run and tested.

Further documentation can be found in the Elektor project folder [3] where all the programs and files used in this article are available.

Figure 4 shows the application running on the Maixduino.

After importing the required modules, the camera and LCD are initialised and configured. The NN is loaded from address 0x200000 and then becomes available as the `task` object. In the `while` loop an image is taken and then displayed on the LCD. The image is then prepared for the NN by using conversion to greyscale, then resizing it to 28 × 28 pixels in size and inverting it (black background) so it appears in the same format as the numbers in the MNIST dataset. After the pixel values have been converted to a format that can be used by the AI algorithm (usually divided by 255), the image is transferred to the NN for recognition. This results in an output in the fmap

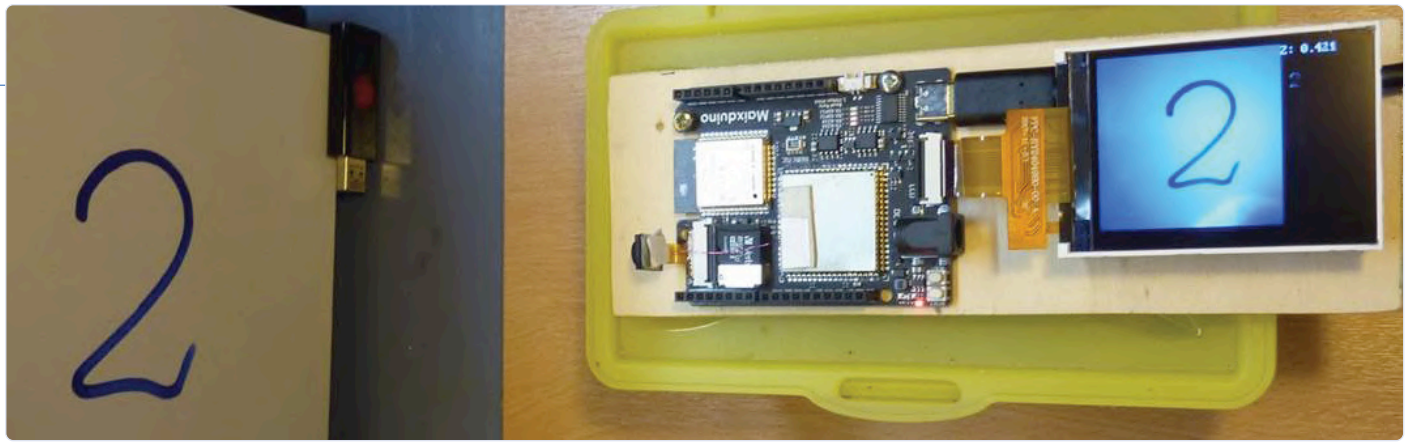


Figure 5. Test setup of number recognition with Maixduino.

variable. Finally, the number recognised, including a value indicating its recognition probability, appears on the display. There we have it: hand written number recognition in just 25 commands!
The practical test (**Figure 5**) confirms that the NN is functioning correctly and, with a recognition rate of a few images per second, demonstrates the performance of the KPU.

The future looks bright!

With its powerful hardware and software development environment, the Maixduino platform is well suited to developers wanting to make a move into the field of Artificial Intelligence. Its low power requirements make it ideal for use in mobile devices, bestowing them with trained NN capabilities. If you plan to develop, train, and execute your NN all on the same platform, it makes sense to invest in a more powerful Linux system such as the Nvidia Jetson Nano. This comes with a good selection of software examples and responsive customer support. Platforms such as the Rock Pi N10 are also worth considering. However, always do your research and check that the software and documentation is of good quality before you invest in something new. A Linux-PC, with or without a GTX graphics card, also helps when exploring this technology space.
Research in AI is currently on a roll. In this short series we have only been able to scratch the surface of the topic of AI. In addition to the method of *Supervised Learning,* which was used here (that is, a NN trained with labelled data), there are many other AI approaches.
In *Unsupervised Learning*, data without any labels are fed into the neural network and divided into clusters, i.e. groups, that the AI determines to contain similar characteristics. This technique is used to identify patterns in large databases of information. Another interesting application is the de-noising (noise reduction) through *autoencoders* for optical character recognition applications.
And how can NNs be better than humans in certain areas, such as beating the best Go players, when the training data is provided by humans? To achieve this, another type of AI, known as *Reinforced Learning*, is used. Here the NN is motivated by a reward system in which several analysis steps are combined to determine the greatest reward (yield). This method is used in the finance sector and for computer games, amongst others.
One weakness of AI implementations at present are there 'black box' architecture, i.e. after data entry, a classification is made without insight as to how it came to that conclusion. It remains unclear, for example, why an AI used in recruitment recommends one candidate over another. In one example it turned out that such a system tended to recommend men simply because the training data had contained more men than women. Intensive research is being carried out on this weak point to create *Explainable AI* so that, in future, NNs will be able to justify their decisions and thus make them comprehensible to their users.

You too can take part and become part of this exciting area of AI development. Take advantage of all the support provided by videos, tutorials, specialist literature, and user groups. And why not add some level of AI capability to your next project? The subject may be complex but, as always, the key to success is to persevere so, as Winston Churchill said, "Never give up!"

200023-C-03

SHOPPING LIST

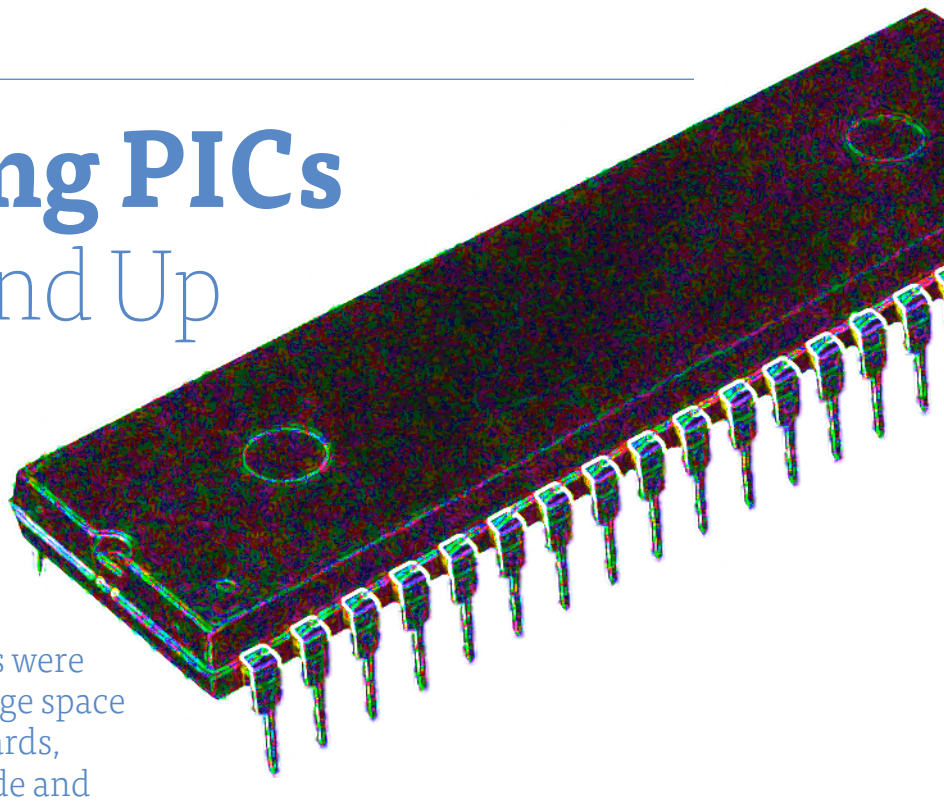
› **Sipeed MAix BiT Kit for RISC-V AI+IoT**
www.elektor.com/sipeed-maix-bit-kit-for-risc-v-ai-iot

WEB LINKS

[1] **AI for Beginners, Elektor May & June 2020**: www.elektormagazine.com/200023-02
[2] **AI for Beginners (2), Elektor July & August 2020**: www.elektormagazine.com/200023-B-04
[3] **Project software**: www.elektormagazine.com/200023-C-01
[4] **ESP32 firmware**: https://github.com/sipeed/Maixduino_esp32_fimware/releases/tag/v1.4.0
[5] **ESP demo program**: https://github.com/sipeed/MaixPy_scripts/tree/master/network

Programming PICs from the Ground Up

Assembler routine to output a sine wave



By **Tam Hanna** (Slovakia)

Back in the day, when 8-bit processors were king, memory chips had limited storage space and were expensive. By today's standards, software development tools were crude and limited. To get any sort of performance from a microprocessor running at less than 5 MHz, the code needed to be efficient. Therefore, programmers often had little choice but to write in assembler (using instructions that have a very strong resemblance to machine code), passing it through a linker to produce an executable hex file.
In this project we write a routine in assembler and run it on an up-to-date PIC processor, outputting a sine wave via its in-built digital-to-analog converter (DAC).
This project should give you a new appreciation of all the unseen work that compilers of higher-level languages normally take care of. However, if you want the best performance from a processor and wish to see what is going on at register level, you have to work in assembler.

For software development we will use the Microchip Technology MPLAB X Integrated Development Environment (IDE) and a PIC 16F18877 as our target device. Programming takes place using the ICSP interface, so you have freedom to choose any development board for this device (a prototyping plug-board will also do).
To produce a continuous 'digitized' sine wave from a DAC we have to supply it at regular intervals with digital values corresponding to voltage levels of a sine wave. Here we use 32 discrete values for every period. Microsoft Excel can be used to generate the `sin()` function values. The Excel table shown in **Figure 1** shows the time intervals from 0 to 31 in the column on the left. The formula `=A2*((2*PI())/32)` provides us the values. The values are for a period from 0 to 2π radians. The third column contains the sine values of the radians in column 2 `=SIN(B2)`. These values are in the range of -1 to +1, but the DAC can only work with positive values. To resolve this, we add a fixed offset to all the values `=1+C2` so that the range of -1 to +1 now is in the range 0 to 2 i.e. all positive. The adjusted values are scaled using `D3*(255/2)` so that they are in the range 0 to 255 which corresponds with the 8-bit input values expected by the DAC. Finally, the values are rounded using `ABRUNDEN(E10)`.

Tables in Data Storage

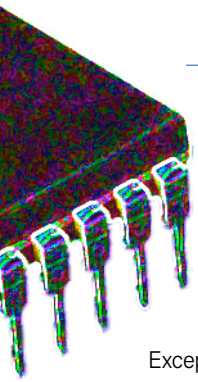
The values corresponding to a sine wave function are constants so they can be stored in a table in memory. To retrieve a value in the table, we can use the `RETLW` instruction that returns with the table value in W (the accumulator).

In assembler we can define an area in memory where the table will be located. In higher-level languages, we are normally not bothered by such details and the compiler will work all this out for itself. We could start the area almost anywhere but, as we will see later, there are some benefits if we start the table at certain addresses. Here we define the start address of the table at `0x200`:

```
TABLE_VECT  CODE  0x0200
  dt  127, 152, 176, 198, 217, 233, 245, 252, 255,
   252, 245, 233, 217, 198, 176, 152, 127, 102, 78,
   56, 37, 21, 9, 2, 0, 2, 9, 21, 37, 56, 78, 102
END
```

The `dt` (define table) directive stores the values provided as a table in memory. This small code fragment can be put through the assembler. The MPLAB assembler ignores many critical situations so when it does throw up a warning such as:

```
Warning[202] C:\USERS\TAMHA\MPLABXPROJECTS\CH6-
   DEMO1.X\NEWPIC_8B_SIMPLE.ASM 72 : Argument out of
   range.  Least significant bits used.
```



it's important to take it seriously and find out what the problem is; mistakes are an essential part of the learning process. How we have defined the table is incorrect, but this makes for a good opportunity to undertake some experimentation.

Assembling and Disassembling

Except when macros are used, there is a direct connection between mnemonics and machine code. Machine code is created by assembling the mnemonics, while the reverse process is performed by a disassembler.

Double -lick the *Usage Symbols disabled* (**Figure 2**) tab at the bottom right of the display. Select the *Load Symbols when programming or building for production* checkbox so that the analysis tools are used in the compilation process.

After clicking on *Apply*, run the compiler again so that the memory usage can be displayed. The option *Window* > *Debugging* > *Output* > *Disassembly Listing File Project* tells the compiler to display a disassembled version of the original code section. This can be useful because it shows us how the compiler has handled the code (**Figure 3**).

The output consists of six columns, with the numbers on the far left describing the 'logical' address of the word in the program memory of the PIC. The next column is the decimal equivalent of the command. The third column contains the disassembled version generated from the hex machine code (for example, constant or variable names and comments in the original assembler file cannot be included). The fourth column specifies the line number in the .asm file and, after the colon, there is the respective line that is responsible for the binary word to the left.

Now we can see that the assembler has by default assumed the values we stored in the table are hexadecimal as it has only used the last two digits of each number (in the 8-bit world, a hex value consists of two characters):

```
0200  3427  RETLW 0x27     73:     dt  127, 152, 176,
   198, 217, 233, 245, 252, 255, 252, . . .
0201  3452  RETLW 0x52
0202  3476  RETLW 0x76
0203  3498  RETLW 0x98
0204  3417  RETLW 0x17
0205  3433  RETLW 0x33
. . .
```

Values entered into such tables can be any of hexadecimal, binary, octal, decimal etc., so it's necessary to use a 'base indicator' symbol before each value. This will let the assembler know that, in this case, the values are decimal. This is done by preceding each value with a decimal point (i.e. a period or full stop). Now, during assembling, the compiler will take the maximum (8-bit) decimal value of 255 and convert it into the maximum hex value FF for use in the code. The table now looks like this:

```
TABLE_VECT  CODE  0x0200 dt  .127, .152, .176, .198,
   .217, .233, .245, .252, .255, .252, .245, .233,
   .217, .198, .176, .152, .127, .102, .78, .56, .37,
   .21, .9, .2, .0, .2, .9, .21, .37, .56, .78, .102
END
```

Schritt	Laufwert	Sinuswert	Bereinigt	DAC-Wert	Abgerundet
0	0,0000	0,0000	1,0000	127,5000	127
1	0,1963	0,1951	1,1951	152,3740	152
2	0,3927	0,3827	1,3827	176,2921	176
3	0,5890	0,5556	1,5556	198,3352	198
4	0,7854	0,7071	1,7071	217,6561	217
5	0,9817	0,8315	1,8315	233,5124	233
6	1,1781	0,9239	1,9239	245,2946	245
7	1,3744	0,9808	1,9808	252,5501	252
8	1,5708	1,0000	2,0000	255,0000	255
9	1,7671	0,9808	1,9808	252,5501	252
10	1,9635	0,9239	1,9239	245,2946	245
11	2,1598	0,8315	1,8315	233,5124	233
12	2,3562	0,7071	1,7071	217,6561	217
13	2,5525	0,5556	1,5556	198,3352	198
14	2,7489	0,3827	1,3827	176,2921	176
15	2,9452	0,1951	1,1951	152,3740	152
16	3,1416	0,0000	1,0000	127,5000	127
17	3,3379	-0,1951	0,8049	102,6260	102
18	3,5343	-0,3827	0,6173	78,7079	78
19	3,7306	-0,5556	0,4444	56,6648	56
20	3,9270	-0,7071	0,2929	37,3439	37
21	4,1233	-0,8315	0,1685	21,4876	21
22	4,3197	-0,9239	0,0761	9,7054	9
23	4,5160	-0,9808	0,0192	2,4499	2
24	4,7124	-1,0000	0,0000	0,0000	0
25	4,9087	-0,9808	0,0192	2,4499	2
26	5,1051	-0,9239	0,0761	9,7054	9
27	5,3014	-0,8315	0,1685	21,4876	21
28	5,4978	-0,7071	0,2929	37,3439	37
29	5,6941	-0,5556	0,4444	56,6648	56
30	5,8905	-0,3827	0,6173	78,7079	78
31	6,0868	-0,1951	0,8049	102,6260	102

Figure 1. The data table is ready for use.

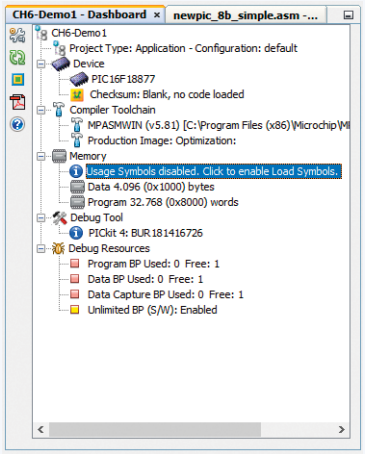


Figure 2. Double-click this line to open the options window.

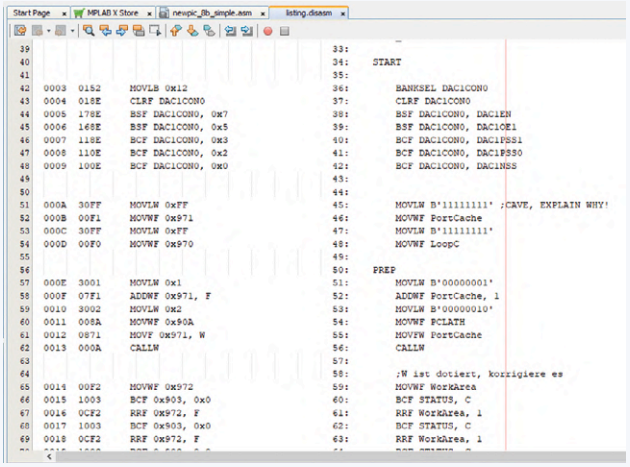


Figure 3. The Disassembled output shows the raw machine code.

CALLW	Subroutine Call With W
Syntax:	[*label*] CALLW
Operands:	None
Operation:	(PC) +1 → TOS, (W) → PC<7:0>, (PCLATH<6:0>) → PC<14:8>
Status Affected:	None
Description:	Subroutine call with W. First, the return address (PC + 1) is pushed onto the return stack. Then, the contents of W is loaded into PC<7:0>, and the contents of PCLATH into PC<14:8>. `CALLW` is a 2-cycle instruction.

Figure. 4. The CALLW mnemonic calculates the address via a comparatively complex process.

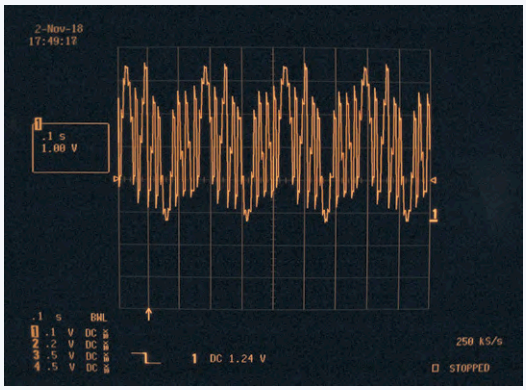


Figure 5. That doesn't look anything like a sine wave.

Accessing Tables

To retrieve information stored in the table, we use the `RETLW` instruction to jump to the table. On returning, the W register (the accumulator) will contain the value from the table. To understand the process we will first convert the start address of the table `0x0200` into its binary equivalent which is `0000 0000 0000 0010 0000 0000 0000 0000`.

Calls to locations in the program memory can be made using the `CALLW` instruction, and **Figure 4** gives more detail on how this instruction works.

The full program memory address space of the PIC contains 32,768 words, which requires a 15-bit address pointer to access the full memory space. Instructions like `CALLW` use the W register to pass the program pointer value (pointing to the location in memory where the subroutine begins). W is only 8-bits long, so it's necessary to make use of the PCLATH register to store the upper value of the program counter proior to the call.

To reduce unnecessary work and possible timing conflicts, the PIC will only use the value of PCLATH if it is required. We can assemble the pointer values in the registers before making the call. Writes to PCLATH leave the current value of the program pointer unaffected.

Initialization of the program begins by incrementing the run value:

```
WORK
    BANKSEL DAC1CON1
    MOVLW B'00000001'
    ADDWF PortCache, 1
```

By checking the full binary address given above we can see what value needs to be loaded to the upper part of the program counter. This value is transferred to PCLATH via the W register using

```
    MOVLW B'00000010'
    MOVWF PCLATH
```

Now we are all set to make the jump. Executing `CALLW` results in the value from the table being returned in the W register. This value is then passed to the DAC to output the corresponding analogue voltage level:

```
    MOVFW PortCache
    CALLW
    MOVWF DAC1CON1
    CALL WAIT
```

Now, if we try executing the program the controller runs until MPLAB stops execution at some point.

The reason is that the table consists of 32 values but the program is working with 255 values. The additional memory locations may contain random values from previously installed firmware. As the pointer increments the program finds itself running into unchartered territory.

This next code section limits the range by subtracting 31, detecting if the zero flag is set by the operation, and then using `CLRF` with `PortCache`:

```
    MOVFW PortCache
    SUBLW .31
    BTFSC STATUS, Z
    CLRF PortCache
```

Finally, we need to jump back to the start of the loop:

```
    GOTO WORK        ; loop forever
```

Now we can run the program. The output shown in **Figure 5** looks pretty chaotic and nothing like the sine wave we were expecting. The reason for this is that the DAC in the processor only accepts values in the range from 0 to 31 but we are supplying values in the range 0 to 255.

Tables from Memory

We could resort to Excel to produce the scaled-down values we require for the sine wave and use them in the table, or we could use the processor to carry out the necessary division of the values in software. However, using assembler we have direct access to binary values stored in registers. The simplest method to carry out a divide-by-2 operation on a binary value is to shift its pattern of ones and zeroes in the register one place to the right. This ignores any carry bits and shifts a zero into the most significant position in the register. To convert values in the range 0 to 255 into the range 0 to 31 we need to divide the value by eight. This corresponds to shifting the value in the register to the right three times.

When the values are copied into memory, we have to calculate target addresses. To do this, we look at the Core Registers. PCLATH is already known, but we are also interested in INDF and FSR. Our PIC has two 16-bit File Select Registers (FSR). These are able to access all file registers and program memory allowing one Data Pointer for all memory. The Indirect File Registers (INDF) are not physical registers. An instruction that accesses an INDFn register accesses the register at the address specified by the File Select Register (FSR). Write operations to the program memory cannot be made via the INDF registers. A special feature of the PIC is that the choice between program and

working memory is made via bit 7 of the FSRxH register. If it is set, the label points to a location in program memory; if not, it addresses data memory.
We start again by storing the variables. This time we need a total of 32 bytes of memory - an allocation size too big to be used as a 'shared' memory facility.
We access memory in a bank to be selected by assembler and work with relative addressing. Additional parameters have not been specified so MPASM grants free choice in the position of the `DataBuffer`:

```
udata_shr
   LoopC res 1
   PortCache res 1
udata
   DataBuffer res 32
```

The high and low parts of the address remain uncertain. Fortunately, the linker makes our work easy with two Operators:

```
START
  MOVLW high DataBuffer
  MOVLW low DataBuffer
```

With this knowledge, we can copy information from the program memory into the main memory. Shift operations do not work directly in W so an additional variable is required:

```
udata_shr
   LoopC res 1
   PortCache res 1
   WorkArea res 1
```

The initial condition is important when working with data tables. We preload PortCache with 1111.1111 because the loop increments *before* it is used. The first pass therefore increments it to 0. If we had preloaded with 0, the first pass would skip 0 and write to location 1:

```
START
. . .
   MOVLW B'11111111'
   MOVWF PortCache
   MOVLW B'11111111'
   MOVWF LoopC
```

The program consists of two loops: the PREP-loop prepares and writes the table of sine wave values to a table, while the `Work` loop sends the values to the DAC so that the signal waveform can be output. PREP begins with an access to the table:

```
PREP
   MOVLW B'00000001'
   ADDWF PortCache, 1
   MOVLW B'00000010'
   MOVWF PCLATH
   MOVFW PortCache
   CALLW
```

Now, with the value in W, we need to move it to the F register and perform three rotate right instructions to divide its value by 8:

```
   MOVWF WorkArea
   BCF STATUS, Z
   RRF WorkArea, 1
   BCF STATUS, Z
   RRF WorkArea, 1
   BCF STATUS, Z
   RRF WorkArea, 1
```

To avoid errors generated by the state of the carry bit in the status register, `BCF STATUS, Z` is used to clear the carry flag in the register before each shift operation. The listing contains two small errors which will prove interesting to resolve.
Now we need to ensure that INDF points to the correct memory location: To achieve this, the registers FSR0H and FSR0L are loaded with address data. The H register (high) contains the higher part and the L register (low) the lower part:

```
   MOVLW high DataBuffer
   MOVWF FSR0H
   MOVLW low DataBuffer
   MOVWF FSR0L
```

INDF0 now points to the beginning of the memory area. We need to add the offset that identifies each location. It is not certain that the beginning of the field is at the beginning of a page. If there was an overflow when adding the offset in the L part, the H part would not notice this. As a solution to this problem, the status of the C bit is checked and the value of FSR0H is incremented if there is an overflow:

```
   MOVFW PortCache
   CLRC
   ADDWF FSR0L
   BTFSC STATUS, C
   INCF FSR0H
```

The `CLRC` (Clear Carry) command is a macro that clears the C bit in the status register. This ensures INDF0 is correctly configured. We have to load and write out the value temporarily stored in the work area:

```
   MOVFW WorkArea
   MOVLW INDF0
```

Finally, we need to make sure that the loop keeps running:

```
   MOVFW PortCache
   SUBLW .31
   BTFSS STATUS, Z
   GOTO PREP
   CLRF PortCache
```

As the code is starting to get more involved and complex, it would be helpful to use some of the IDE tools to examine program execution more closely and verify correct operation.

Troubleshooting in Assembler

Placing breakpoints is theoretically easy; click on the line numbers in

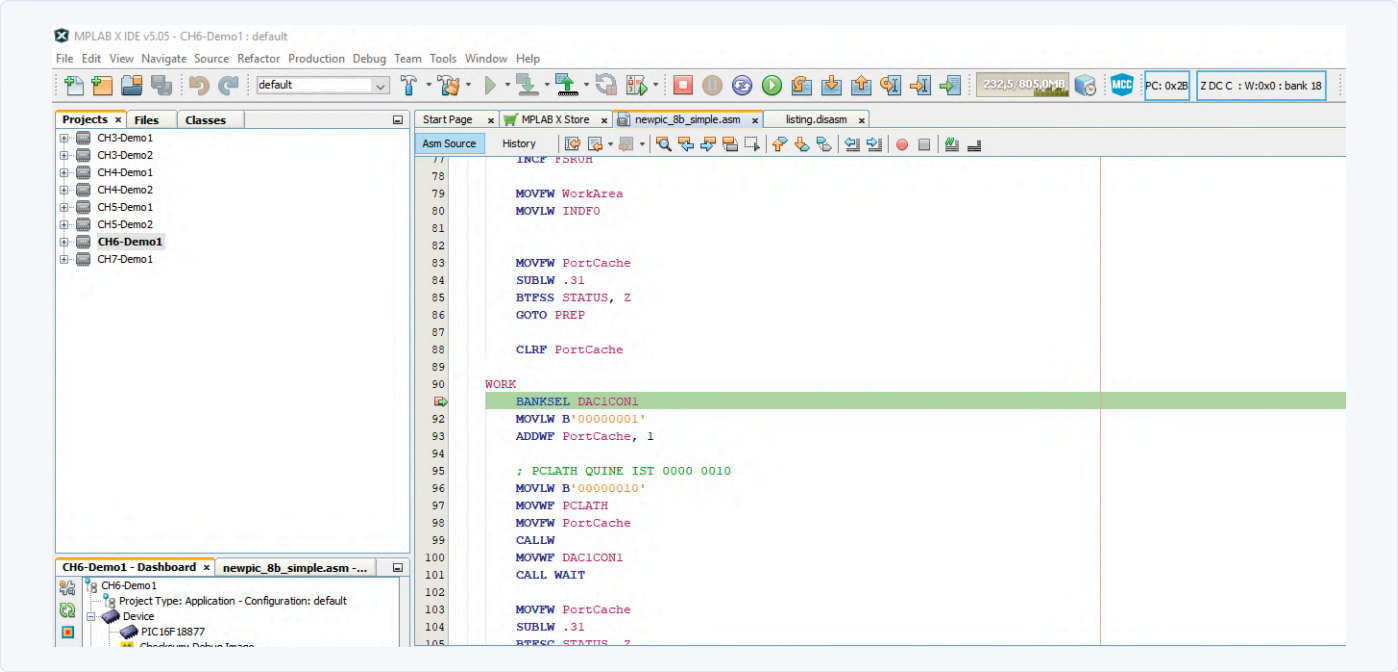


Figure 6. The debugger has interrupted program execution.

the IDE to place a red stop symbol. However, since the microcontroller can only manage one hardware breakpoint, this causes a message to appears. MPLAB uses up debugging resources to enable it to run in stages. We do not require breakpoint emulation at the moment, so we can remove the message by clicking *No*. As the PIC supports software breakpoints, you can affirm with *Yes*.

As long as you place more than one breakpoint in the assembler file, Microchip activates this function automatically.

Our PIC will only support one hardware breakpoint. Click on the down arrow next to the debugger symbol in the toolbar and select the *Debug main project* option. MPLAB opens a disassembly window that we can then close. After reaching the breakpoint, the IDE shows the status as in **Figure 6**.

The line with the green background and right-pointing arrow in the line number column on the left is the current instruction. Symbols in the toolbar, such as stop and jump, allow user interaction with program. *Window Target Memory Views File Registers* opens a window to view

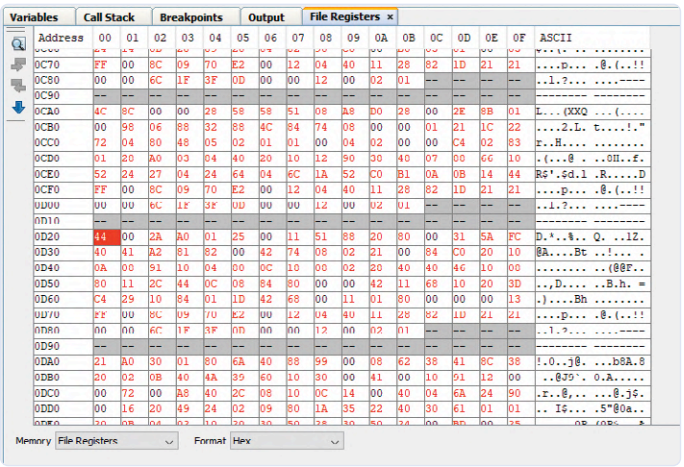


Figure 7. These values don't look correct.

the PIC memory space. Place a cursor like the mouse pointer over the DataBuffer Declaration. This opens a tooltip window with the address of the first byte and its value. On the author's workstation this position is `0xD20`.

It is more convenient to click the blue arrow (*GoTo*) in the File Registers window. In the *GoTo What* box, select *Symbol* and select *DataBuffer*. Close the popup after clicking the *GoTo* button to see the result shown in **Figure 7**. The red highlighted box is the first byte of the table. It is obvious there is an error because other values shown in the table such as FC are outside the permitted range.

To investigate we can fill the memory area with an easily recognizable bit pattern. A good example would be to write the value 11111111 into the indirect register:

```
MOVFW PortCache
CLRZ
ADDWF FSR0L
BTFSC STATUS, C
INCF FSR0H
MOVFW B'11111111'
MOVLW INDF0
```

If you view the code in the debugger, you will see a sequence of the same values. In most cases, they should not have the value FF. The code has a small error. We can use the `MOVLW` instruction to load the value of a memory location into the W register:

```
MOVFW B'11111111'
MOVLW INDF0
```

From MPLAB's point of view, the literal INDF0 is a number: after compilation, values like PORTA are just numbers. So our program copies the address of the register into all 32 memory locations. Nevertheless, we are one step further since we have already verified the address data. A corrected version of the program now looks like this:

```
MOVLW B'11111111'
MOVWF INDF0
```

Now that the memory address calculation works, we can eradicate the actual calculation error. The first problem was using the `MOVLW` instead of the `MOVWF` instruction, which made writing to INDF impossible:

```
MOVFW WorkArea
MOVWF INDF0
MOVFW PortCache
SUBLW .31
BTFSS STATUS, Z
GOTO PREP
```

When looking at the output, we can see that the values are not correct. The cause of this problem is that the `RRF` instruction can set the carry bit. Our code however has only taken care of the Z flag; we need to make a small correction:

```
MOVWF WorkArea
BCF STATUS, C
RRF WorkArea, 1
BCF STATUS, C
RRF WorkArea, 1
BCF STATUS, C
RRF WorkArea, 1
```

The program is now ready to run and the table of sine wave values appear in the debugger window. For completion, we have to ensure in the working loop that the values are taken from the data memory. This requires an increment of the run variable in order to generate a continuous index:

```
WORK
    BANKSEL DAC1CON1
    MOVLW B'00000001'
    ADDWF PortCache, 1
```

Indirect addressing is suitable for reading and writing. We load the two parts of the address of the buffer in FSR0H and FSR0L. Then we add the offset and check for an overflow. If an overflow occurs, we increment the upper register:

```
MOVLW high DataBuffer
MOVWF FSR0H
MOVLW low DataBuffer
MOVWF FSR0L
MOVFW PortCache
CLRZ
ADDWF FSR0L
BTFSC STATUS, C
INCF FSR0H
```

What is new is that we are reading from register INDF0. The value is loaded into the DAC1CON1 register of the DAC:

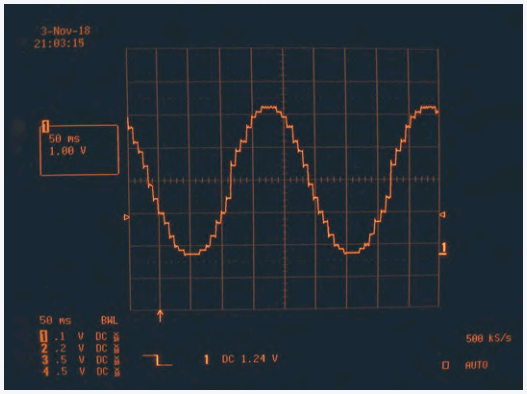


Figure 8. The output signal from the DAC approximates to a 4 Hz sine wave.

```
MOVFW INDF0
MOVWF DAC1CON1
CALL WAIT
```

The rest of the program is an ordinary loop that, among other things, takes care of the increment operation:

```
MOVFW PortCache
SUBLW .31
BTFSC STATUS, Z
CLRF PortCache
GOTO WORK
```

With that the program is finished and you can see the resultant sine wave output signal in **Figure 8**.

Conclusion

This project shows that interesting experiments can still be implemented with 8-bit microcontrollers. More information on this topic can be found in my new book *Microcontroller Basics with PIC*.
If you enjoyed this article, please let me know. I always welcome constructive feedback!

200154-02

IKEA Lamp Hack

Equipping an inexpensive IKEA lamp with NeoPixel LEDs and WLAN

By **Hans Henrik Skovgaard** (Denmark)

You can of course buy remotely controlled, Wi-Fi-connected multi-color lamps from several suppliers. Why would you want to build one yourself? Well, it can work out cheaper, you can customize it to your heart's content, and you will get a sense of satisfaction and achievement that you'll never get from an off-the-shelf unit.

As an engineer, I like working out solutions to problems; that's what motivates me. Some time ago, I faced an unexpected challenge when my son gave me a NeoPixel Jewel 7 board [1] from Adafruit. It was a leftover from a project he'd been working on at university. This small, circular circuit board can be seen in **Figure 1**. The board is fitted with seven programmable NeoPixel RBG LEDs in and SMD outline. My only thought was, "What on Earth can I use it for?"

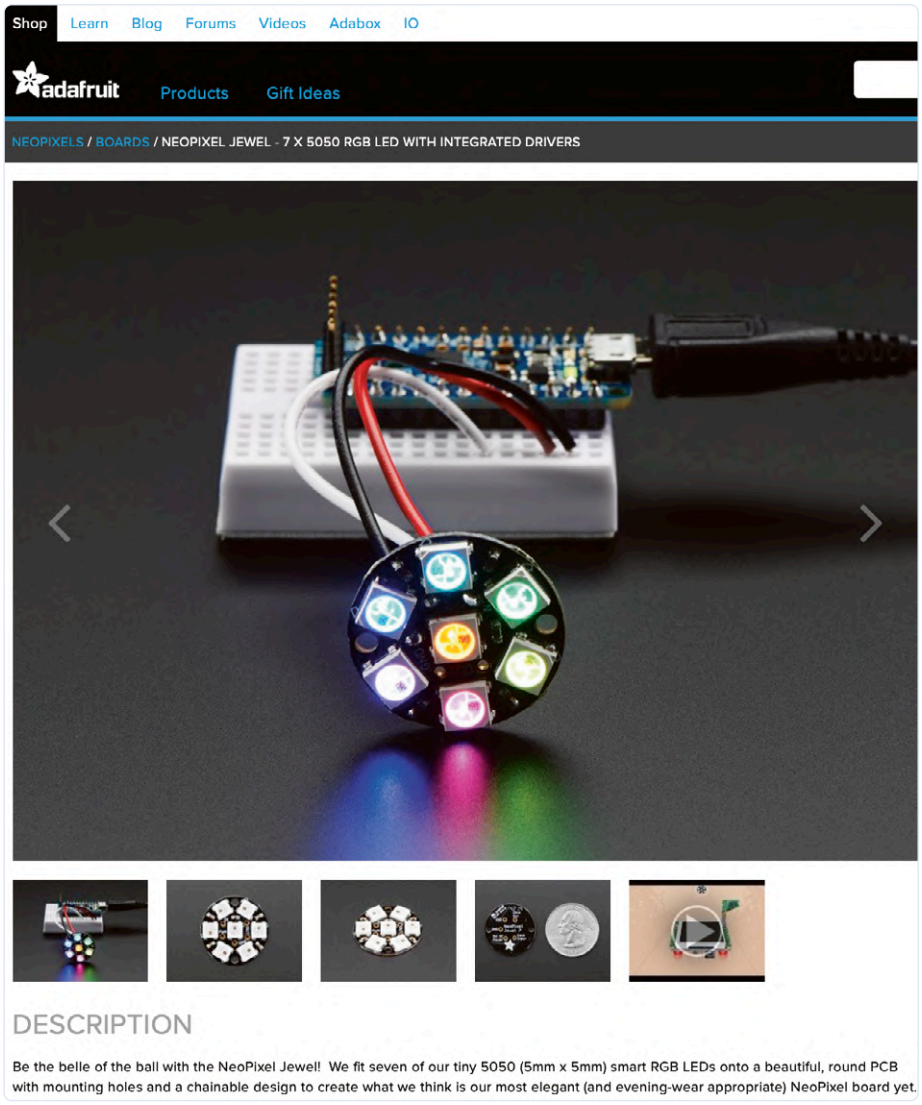


Figure 1: NeoPixel Jewel 7 connected to an Arduino board **[1]**.

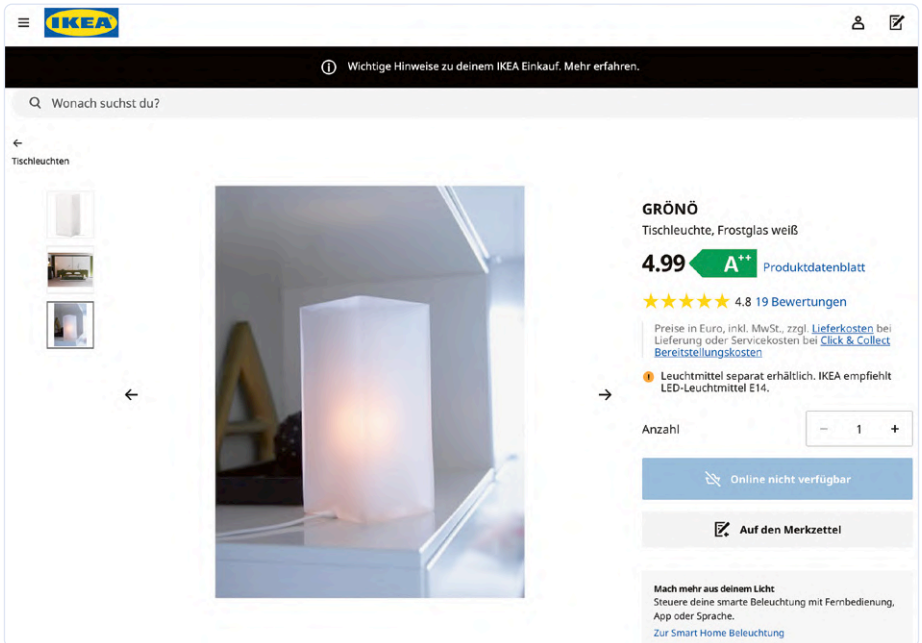


Figure 2: The low-cost IKEA Grönö lamp **[4]**.

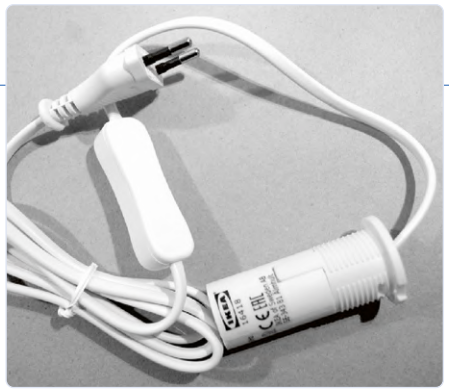


Figure 3: The IKEA lamp before hacking.

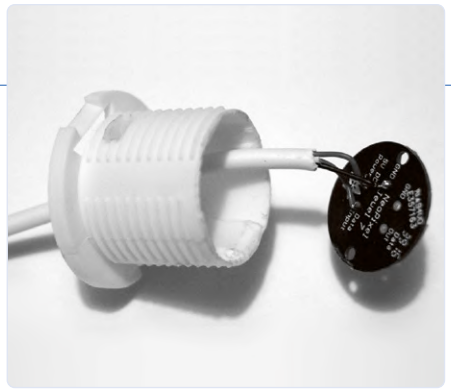


Figure 4: The LED board attached atop the modified lamp holder.

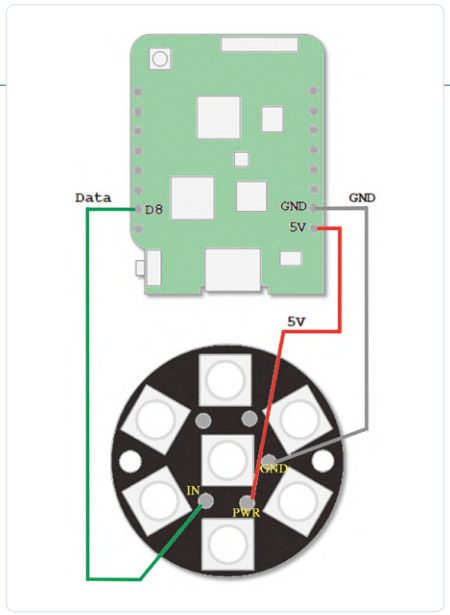


Figure 5: The hack circuit is very simple: microcontroller board + LED board.

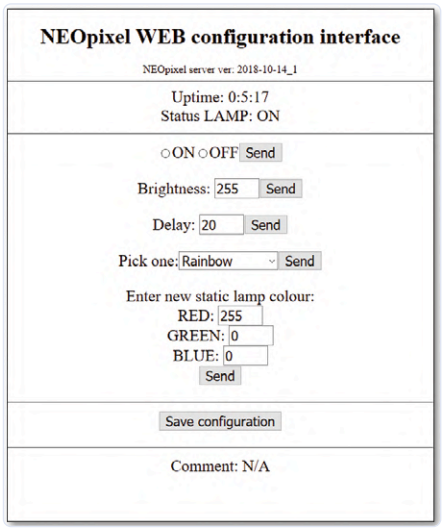


Figure 6: Simple configuration interface of the web server.

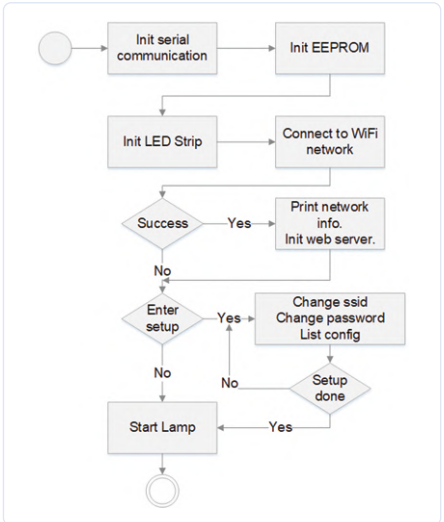


Figure 7: Flow diagram of the setup process.

NeoPixel + Arduino

WorldSemi Co. has been supplying RGB LEDs with an integrated controller in its product range for some time now. These feature a three-wire interface, allowing them to be easily "daisy-chained" for larger display applications. Each RGB LED can be addressed individually via a single serial data signal. You will get lots of hits, mostly from the Far East, when you enter the terms NeoPixel or WS2812B on eBay. Since I had already developed a whole series of Arduino projects, I knew it would not be difficult to control these LEDs with the help of the appropriate library from Adafruit [2]. My first impressions of these LEDs were recorded on my retina. They are quite bright at full power, so please be careful not to stare at them directly!

The Arduino boards are a good general-purpose controller solution for many experimental setups. But, if you need an Internet connection, it is better to use a different solution. At the time I was working with an ESP8266 MCU from Espressif in the form of the 'D1 mini Pro' from WeMos [3]. It seemed to me that this board, together with the NeoPixel LED board, would make for a good combination. To make sure the finished light source blended well into a domestic environment, it would be necessary to build it into a suitable housing.

Lamp tweaking

My solution to the problem was to modify a standard IKEA Grönö [4] lamp for the housing. This lamp (**Figure 2**) only costs € 6.50 in Denmark (£7 in the UK), making it an ideal candidate for hacking. Should I end up destroying it in the process, then I haven't really lost much — this should always be your number one consideration before you set about hacking anything.

Figure 3 shows the lamp internals; it's just an E14 style lamp holder and power cord with an in-line power switch. We will only need three wires to connect the ESP8266 to the NeoPixel board. The lamp holder base can be modified and the power cord discarded. The board is then fixed on top of the cut-down lamp holder base with glue. Finally, the power cord is replaced by three short leads (**Figure 4**).

The lamp controller circuit (**Figure 5**) could hardly be simpler. In addition to the NeoPixel board and the ESP8266, the lamp requires a 5 V, 500 mA mains adapter power supply with a micro USB plug. The rest of the project is purely software. The Neopixel board is fitted with seven RGB LEDs, each with its own in-built controller in SMD 5050 format. You can also buy these LEDs individually under the designation WS2812B [5] and thereby make almost any complex string or matrix arrangement. Each RGB LED takes up to 60 mA at full brightness, delivering a maximum luminous flux of 20 lm. You can hook them up in series and address them individually by shifting the RGB brightness data serially into the chain at one end. This makes them very easy to control.

Software

An important attribute of the WeMos D1 mini Pro board is that it is supported by the Arduino IDE. Using the Arduino IDE has its advantages and disadvantages — what I find particularly relevant is that it allows me to get a project up and running quickly and without too much fuss. Before we start with the software for this IKEA hack, it is necessary to do a little preparatory work:

- Install the Arduino IDE [6].
- Include support for the ESP8266 MCU [7] in the IDE.
- Install the Adafruit-NeoPixel library within the Arduino IDE [2].
- Install the Arduino ESP8266 file system uploader (for SPI flash) [8].

The software [12] may seem a little complex given the simple task of controlling the LEDs, but the reason for the complexity is that an ESP8266 has built-in Wi-Fi support. If we want to make use of it, we will need some code. The software provides the following capabilities:

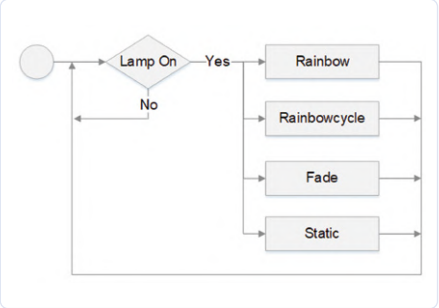


Figure 8: Flow diagram of the mode selection.

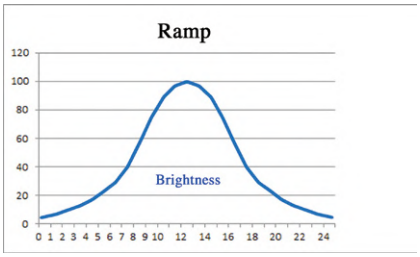


Figure 9: Optimized brightness curve for fading.

- Creation of special lighting effects.
- Internet connectivity via Wi-Fi.
- Web server for lamp configuration.

The lamp can also be operated offline or in standalone mode. In this case, it will run through the colors of the rainbow at a predefined rate (e.g., 20 ms for each of the 8x8x8 color levels). If you want to be able to configure the lamp during operation, you must activate the Wi-Fi link so that the NeoPixel lamp can communicate via your local network. The necessary parameters for the Wi-Fi can be hard coded in the source code. This means any changes to the network parameters require a recompilation of the WeMos firmware. Otherwise, they can be defined during the lamp power-on 'boot' period. Once the lamp is connected to the Wi-Fi network, it will be possible to carry out the following operations:

- Turn the lamp on and off.
- Control the brightness.
- Change the delay time between color cycling.
- Select color cycles (currently rainbow, rainbow cycles, static, candle flicker, and fading).
- Set a fixed lamp color.

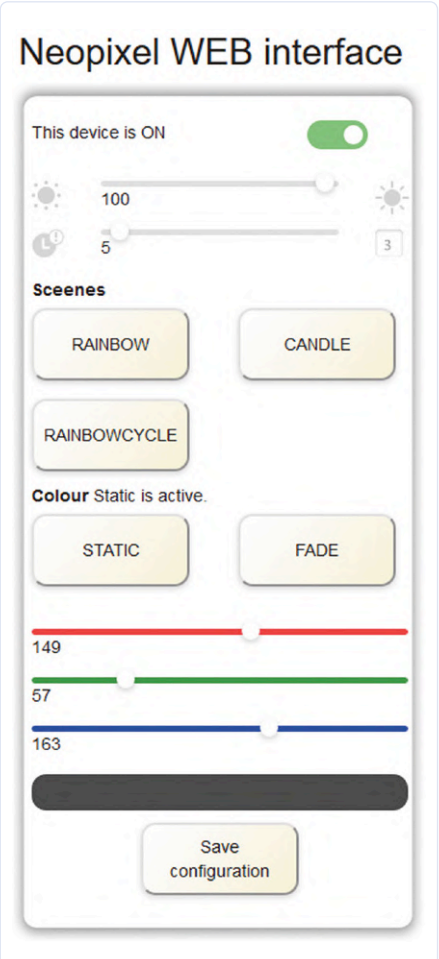


Figure 10: A slicker web interface.

These operations can be performed via a very simple web interface (**Figure 6**). The HTML code for the web page is generated using the `getPage(string str)` function.
All of this is handled in the software and, because of this, is not so easy to make changes to. However, it works well enough for our purposes. The web configuration interface updates the NeoPixel lamp via an HTTP POST request method. The basic procedure is described in my book *IoT Home Hacks with ESP8266* [9]. The software is structured in much the same way as for other similar projects I've developed. **Figure 7** shows the software flow chart during setup with **Figure 8** displaying the same for operation or mode selection.
I quickly discovered that the fade function (increasing and decreasing brightness) requires a little more consideration. If you control the lamp brightness using values that correspond to a linear ramp function, the change in brightness is not satisfying. As shown in **Figure 9**, it's better to use a function that approximates a sine wave to produce a more gentle change in brightness. This brightness curve was simulated in Excel and the values implemented as an array `byte fadeInterpolation[]`. These values can of course be changed.
The article "LED-Dimmers" in *Elektor* 9/2018 [10] describes this effect. The relation between the actual change in a physical stimulus and the perceived change is a psychophysical property characterized by the Weber-Fechner law [11].

Using the Lamp

As soon as the programmed WeMos board is powered on, the software will boot up and enter its switch-on cycle.

Wi-Fi connection (blue light)

In this phase, a connection with the Wi-Fi network is attempted using the network parameters programmed in software. During this phase, each of the LEDs will flash blue one after another.

Wait (red light)

After attempting to connect with the Wi-Fi (successfully or unsuccessfully), the board will look for user interaction via the USB port. During this period, the LEDs will flash red one after another. When an interaction is detected, all the LEDs will go off. If after ten seconds

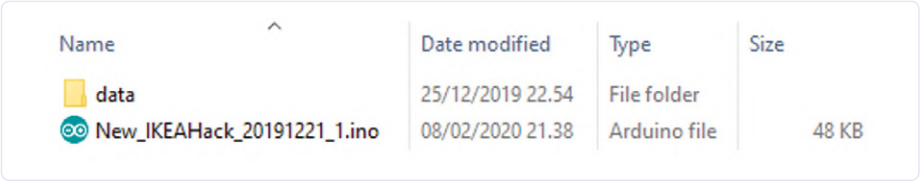
no interaction is detected, the lamp begins running its pre-configured lighting pattern. After switch-on and with an active Wi-Fi connection, it is possible to control the lighting effect or operating mode of the NeoPixel lamp.

A different web interface

Once I had got this far with the project, I showed the results of my efforts to my son. He has a master's degree in interactive design and was not at all impressed with the design of the web interface. He went on to develop a new design (**Figure 10**) that proved difficult to implement in "hard" code.

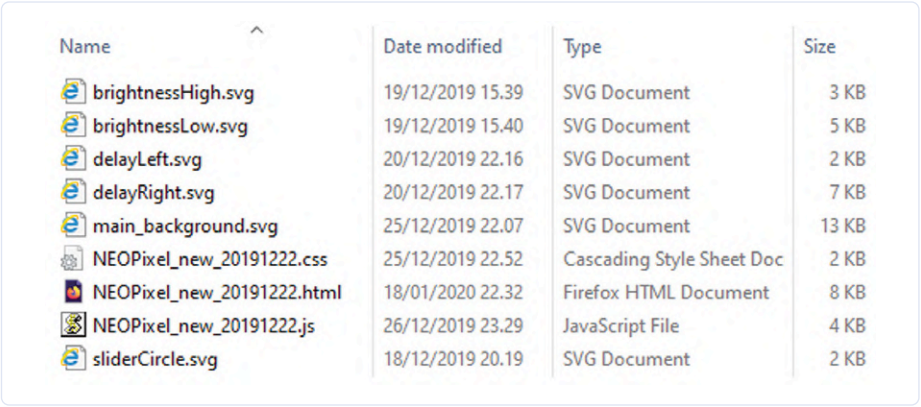
Instead, a design using HTML, CSS, and Javascript was developed. This is where the SPI flash file system comes into play. The ESP8266 MCU provides at least 14 MB of flash memory that can be used via the SPIFFS file system. In my book [9] you will find information on:

- Uploading files to SPIFFS.
- Storing files in a PC for uploading.
- Installation of the necessary software for the Arduino IDE.



Name	Date modified	Type	Size
data	25/12/2019 22.54	File folder	
New_IKEAHack_20191221_1.ino	08/02/2020 21.38	Arduino file	48 KB

Figure 11: The Arduino software directory should look like this.



Name	Date modified	Type	Size
brightnessHigh.svg	19/12/2019 15.39	SVG Document	3 KB
brightnessLow.svg	19/12/2019 15.40	SVG Document	5 KB
delayLeft.svg	20/12/2019 22.16	SVG Document	2 KB
delayRight.svg	20/12/2019 22.17	SVG Document	7 KB
main_background.svg	25/12/2019 22.07	SVG Document	13 KB
NEOPixel_new_20191222.css	25/12/2019 22.52	Cascading Style Sheet Doc	2 KB
NEOPixel_new_20191222.html	18/01/2020 22.32	Firefox HTML Document	8 KB
NEOPixel_new_20191222.js	26/12/2019 23.29	JavaScript File	4 KB
sliderCircle.svg	18/12/2019 20.19	SVG Document	2 KB

*Figure 12: Contents of the **data** directory.*

If you've installed it correctly, your Arduino software directory should look similar to **Figure 11**. **Figure 12** shows the files for the web interface contained in the ***data*** folder. The latest version of the software can be downloaded free of charge from the Elektor website for this article [13]. You will notice that the *NEOPixel_new_20191222_load.js* file is not listed in the ***data*** directory. These functions are coded directly in the ESP8266 software. This allows the ESP8266 MCU to configure the Web interface upon power-up using the parameters stored in the EEPROM.

The web interface concept has been designed for smartphone screens in portrait-display orientation. Since the NeoPixel lamp can be configured by sending it HTTP POST requests, it is also possible to control the lamp via the OpenHAB home automation solution — but that's another story.

Working with the NeoPixel Jewel board has led me to explore many different areas that I wasn't familiar with before the project started.

Overall the result is quite satisfying; this IKEA lamp hack with its shoe-horned NeoPixel board has been lighting up my living room now for over a year.

200165-02

WEB LINKS

[1] NeoPixel Jewel 7: www.adafruit.com/product/2226

[2] NeoPixel library guide: https://learn.adafruit.com/adafruit-neopixel-uberguide/arduino-library-use

[3] Wemos D1 mini Pro: www.wemos.cc/en/latest/d1/d1_mini_pro.html

[4] Grönö lamp from Ikea: www.ikea.com/nl/en/p/groenoe-table-lamp-frosted-glass-white-20373225/

[5] WS2812B data sheet: www.world-semi.com/DownLoadFile/111

[6] Arduino-IDE: www.arduino.cc/en/Main/Software

[7] ESP8266 support: https://arduino.esp8266.com/stable/package_esp8266com_index.json

[8] Uploader for file system: https://github.com/esp8266/arduino-esp8266fs-plugin#arduino-esp8266-filesystem-uploader

[9] *IoT Home Hacks with ESP8266* book: www.elektor.com/iot-home-hacks-with-esp8266-e-book

[10] LED Dimmer article: www.elektormagazine.com/magazine/elektor-60/41945/

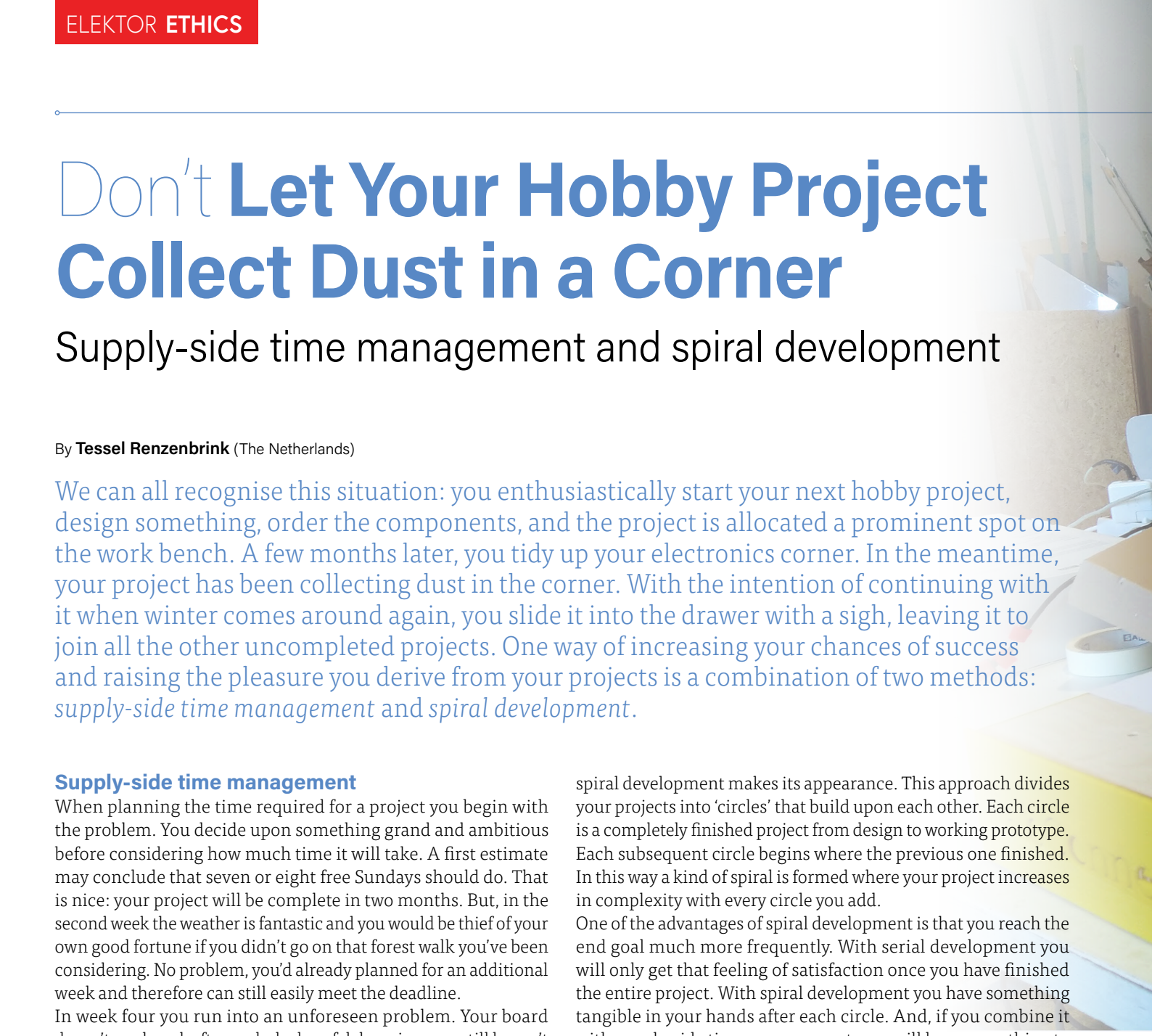
[11] Weber-Fechner law: https://en.wikipedia.org/wiki/Weber%E2%80%93Fechner_law

[12] Software: www.elektor.com/amfile/file/download/file/2134/product/9476/

[13] Weblink to this article (latest software): www.elektormagazine.com/200165-02

Don't **Let Your Hobby Project Collect Dust in a Corner**

Supply-side time management and spiral development



By **Tessel Renzenbrink** (The Netherlands)

We can all recognise this situation: you enthusiastically start your next hobby project, design something, order the components, and the project is allocated a prominent spot on the work bench. A few months later, you tidy up your electronics corner. In the meantime, your project has been collecting dust in the corner. With the intention of continuing with it when winter comes around again, you slide it into the drawer with a sigh, leaving it to join all the other uncompleted projects. One way of increasing your chances of success and raising the pleasure you derive from your projects is a combination of two methods: *supply-side time management* and *spiral development*.

Supply-side time management

When planning the time required for a project you begin with the problem. You decide upon something grand and ambitious before considering how much time it will take. A first estimate may conclude that seven or eight free Sundays should do. That is nice: your project will be complete in two months. But, in the second week the weather is fantastic and you would be thief of your own good fortune if you didn't go on that forest walk you've been considering. No problem, you'd already planned for an additional week and therefore can still easily meet the deadline.

In week four you run into an unforeseen problem. Your board doesn't work and, after a whole day of debugging, you still haven't made any progress. The fun has vanished and, by week five, you have no motivation to continue. Your enthusiasm has now decreased significantly while the first real results seem increasingly out of reach. This is the road that inevitably leads to a drawer or cupboard of half-finished projects.

This example illustrates how demand-side time management works. You let the project determine how much time it requires and you go looking for the hours to make it happen. The reverse of that is supply-side side management. Here you determine how much time you have available and you size your project accordingly. You know for sure that you have time for your hobby next Sunday, so you get to work on a project that you can finish in one Sunday. You divide your Sunday into small chunks for each phase of the project. Two hours for the design, three hours for prototyping, etc. The trick is to stick to the plan. In this way you avoid spending the entire Sunday stuck in the design phase because there is one little detail that you can't quite get right.

Spiral development

Planning on the basis of how much time you have available does not mean that you can only tackle short-duration projects. This is where spiral development makes its appearance. This approach divides your projects into 'circles' that build upon each other. Each circle is a completely finished project from design to working prototype. Each subsequent circle begins where the previous one finished. In this way a kind of spiral is formed where your project increases in complexity with every circle you add.

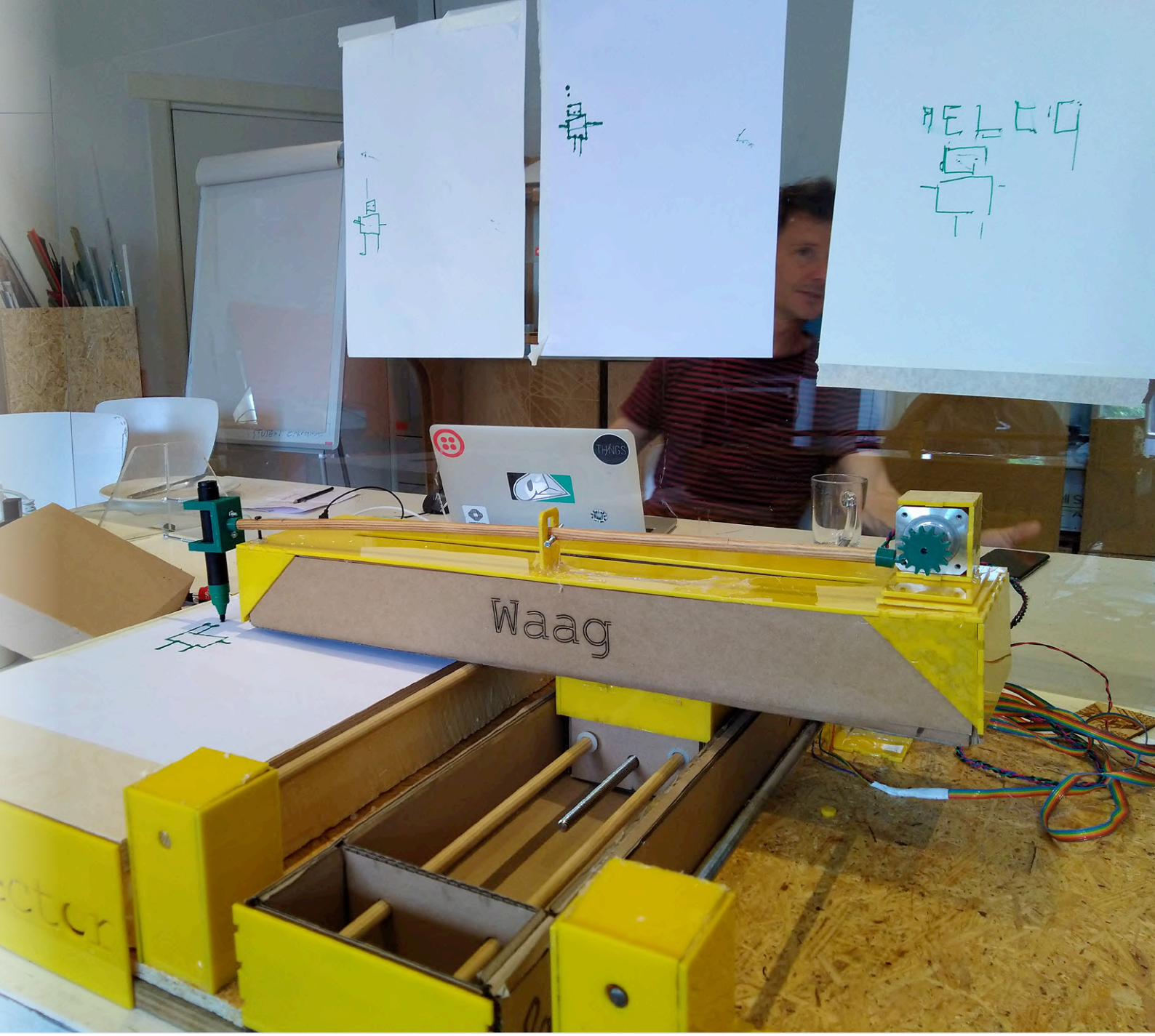
One of the advantages of spiral development is that you reach the end goal much more frequently. With serial development you will only get that feeling of satisfaction once you have finished the entire project. With spiral development you have something tangible in your hands after each circle. And, if you combine it with supply-side time management, you will have something to celebrate every Sunday.

A second advantage is that you test whether your project actually works at the end of each circle. In this way you prevent reaching the end of a long-running project only to discover that the thing doesn't work. Additionally, this makes the debugging of a small project much more manageable than that of a large and complex project.

Building a machine in seven days

We employed these methods when my fellow students and I had to build a machine in one week. We undertook a *Fab Academy* course where computer-controlled manufacturing processes, such as 3D-printing, and electronics production is taught at a killer pace. For the group project we built a remote-controlled drawing machine. Our fellow student in France can now draw cats and robots that appear on paper in our Amsterdam laboratory. To achieve this in seven days, we used supply-side time management and spiral development.

The time pressure showed its positive impact right from the beginning. When five people are working on a project, quite a bit of time can be wasted on deliberation and disagreement of what to make or what the requirements should be. But, because of the unrelent-



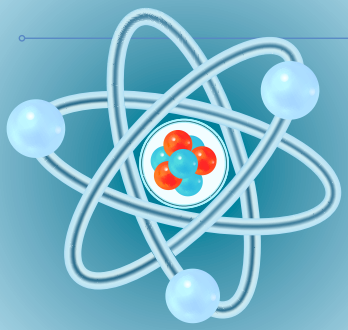
The drawing machine that can be controlled over the internet (photo: Tessel Renzenbrink).

ing ticking of the clock, we quickly found agreement. The goal of the first spiral was the local control of the stepping motors via the serial monitor of the Arduino IDE, and building a simple housing to guide the X and Y axes.

This first phase took longer than we anticipated. To ensure that we remained within the allotted time, we applied 'triage'. This is a method of setting priorities that was first used in field hospitals. Because of a limited treatment capacity, patients were divided into three categories: 'untreatable', 'treat immediately' and 'can wait a little'. We undertook the same with our project. The patient that was mortally wounded was the addition of limit switches for the stepping motors. This seemed to be a simple task but, after a few hours of internet forums and fast-forwarding through Youtube tutorials, we had made no progress. The skill is then not to hold on, but to let go. You can choose to leave that requirement out, or push it out to a later spiral.

The number of shortcuts and 'creative' solutions increased as the finish line came into view. While we made nice finger joints in the early stages, towards the end we simply reached for the glue gun. The final phase of a spiral is exciting because you achieve a lot in a short time. Working with supply-side time management and spiral development makes projects much more interesting because they force you to continue. It is less frustrating because you don't have the time to linger on something that won't succeed. This does not mean that you have to rush through your project, but it does help when setting your priorities. For our overall goal, that is demonstrating a working machine in class, limit switches were not crucial. If you were planning to make the drawing machine available for use by others, then it definitely would be a good idea to include them. So, if that is the case, you can simply plan to add them in a subsequent spiral.

200300-03



Starting Out in Electronics (4)

Easier than imagined!

By **Eric Bogers** (Elektor Netherlands)

Where were we before Corona turned our lives upside down and we treated you to our special summer edition? Precisely: series and parallel connected resistors. We concluded the previous episode with a brain teaser: the H-circuit…

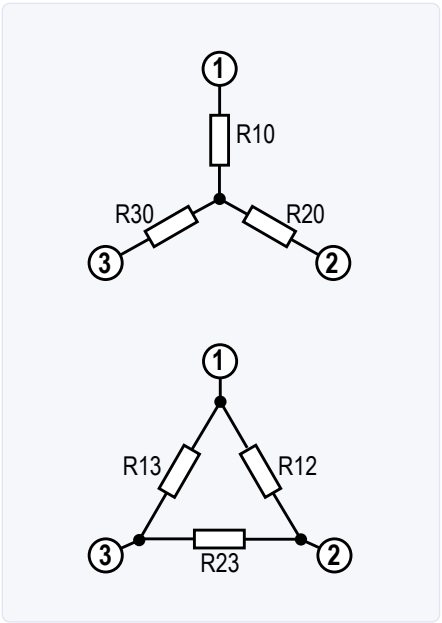


Figure 1: The star to delta transformation.

The main topic for this episode is the star to delta transformation (which can, of course, be considered in the opposite direction: delta to star transformation). The core principle is sketched in **Figure 1**.

The objective here is to convert the star circuit to a delta circuit such that both circuits behave exactly the same from the perspective of the 'outside world.' (The opposite is, of course, also possible.)

For the conversion from star to delta, the equations below apply (we will not bother you with the derivation of these – after all, this is an electronics magazine and not a mathematics magazine):

$$R12 = \frac{R10 \cdot R20}{R30} + R10 + R20$$

$$R23 = \frac{R20 \cdot R30}{R10} + R20 + R30$$

$$R13 = \frac{R10 \cdot R30}{R20} + R10 + R30$$

And the following for conversion in the opposite direction:

$$R10 = \frac{R12 \cdot R13}{R12 + R23 + R13}$$

$$R20 = \frac{R12 \cdot R23}{R12 + R23 + R13}$$

$$R30 = \frac{R23 \cdot R13}{R12 + R23 + R13}$$

Make sure to take careful note of the numbering of the various resistors!

With that clarified, we can now tackle that H-circuit – see **Figure 2**. If we look carefully, we see a delta circuit that has two more resistors connected to its base (R4 and R5). We apply the transformation to the delta comprising of R12, R13 and R23. Let us assume the following resistance values: R13 = 10 Ω, R12 = 20 Ω, R23 = 30 Ω, R4 = 40 Ω and R5 =50 Ω.

When we write out the transformation, we find:

$$R10 = \frac{R12 \cdot R13}{R13 + R12 + R23} = \frac{200}{60}\Omega = 3{,}3\Omega$$

$$R20 = \frac{R12 \cdot R23}{R13 + R12 + R23} = \frac{600}{60}\Omega = 10\Omega$$

$$R30 = \frac{R23 \cdot R13}{R13 + R12 + R23} = \frac{300}{60}\Omega = 5\Omega$$

What remains is a simple combination of series and parallel circuits. We confidently leave it to you to work this out to its conclusion (the result, by the way, is 29.05 Ω).

The approximate approach

Often, much of this computational effort can be saved by little tricks. How? One way is by taking a looki at two 'extreme' situations and applying some thought.

Let's return to our H-circuit from **Figure 2** above. If we mentally remove resistor R23 and put nothing in its place, the total resistance of the network has to increase. (Think: if in a network of resistors we increase the value of a resistor, then the total resistance will *never* be smaller – and here we increased the value of R23 to infinity).

The total resistance of the network is now easily calculated; if you do that yourself (with the same resistance values as in the example) you should arrive at 29.16 Ω.

Now let's replace R23 with the other extreme: a wire link (we therefore reduce the resistor value to zero ohms). Now think again: if in a network of resistors we reduce the value of a resistor, the total resistance of that network is *never* greater. The resulting calculation is simple and the result is 28.8 Ω.

So, that is nice! Irrespective of the actual value of R23, the equivalent replacement resistor for our H-circuit has to be between these two extreme values. And when (as in

this example) these values don't differ more than a percent or two, we can, in practice, work with the average of these extreme values. *Homines perfectici* may calculate everything to the nth decimal place, but that is not often required because the resulting error is generally 'obscured' by the tolerances of the components.

Nevertheless, in other cases the difference can certainly be large and this "approximate" approach then has to be used with caution!

Interconnection resistance

Before we (finally) get to AC voltages, we really need to talk briefly about interconnection resistance. Up until now we have ignored the resistance of the connections. However, every cable and printed circuit board trace has a certain resistance and cannot be ignored in every circumstance!
For the resistance of the interconnect:

$$R = \frac{l \cdot \rho}{A} = \frac{l}{A \cdot \gamma}$$

where ρ is the resistivity and γ the conductivity. These are both material constants. For typical conductors (copper, aluminium, silver) it is a little easier to use the resistivity calculation.

Resistivity and conductivity

material	ρ (Ω·mm²/m)	γ (m/Ω·mm²)
copper	0.017	56
aluminium	0.0287	34.8
silver	0.016	62

Two other variables appear in the formula for conductivity: the length, *l*, of the conductor and the area of its cross-section, *A*.
Let's work this out for a cable reel with 50 metres of cable and a cross-sectional area of 1.5 mm². Such a cable has two conductors, so we have to factor in twice the length:

$$R = \frac{l}{A \cdot \gamma} = \frac{2 \cdot 50m}{1.5mm^2 \cdot 56m/\Omega^2} = 1.19\Omega$$

Now, let's run a current of 16 A through this cable; the voltage drop across the cable is then

$V = R \times I$ = 1.19 Ω × 16 A = 19 V

However, this is not the complete story. The resistivity of a material such as copper is sensitive to temperature and, with increasing temperature, the resistivity increases. The following holds:

$$\rho_T = \rho_{20}\left(1 + \alpha\left(T - 20°\right)\right)$$

where α is the temperature coefficient, which for copper has the value 0.0038, and *T* is the temperature in °C. The value of ρ is always specified at 20 °C.

It can become dangerous...

Suppose that we use our cable reel on a warm summer's day. The insulation becomes quite warm in the sun and the current of 16 A is a significant flow of energy. Let's cautiously assume that the copper conductor reaches a temperature of 60 °C. If we work with those numbers we arrive at a voltage drop of no less than 23.3 V. (Feel free to check that!)
At normal room temperature the cable already dissipates a considerable amount of power (which is all turned into heat):

$P_{loss} = I^2 \times R$ = (16 A)² × 1.19 Ω = 304.6 W

This is quite a lot and the sign of the temperature coefficient (positive) works to our disadvantage. Such a heavily loaded cable becomes warm, which increases the resistivity, which increases the interconnect resistance, which increases the dissipation and, therefore, the cable becomes even warmer and... As you can see, it has a positive feedback effect that can easily lead to overheating and a fire. You have been warned!

200320-04

The magazine article series "Starting Out in Electronics" is based on the book *Electronics Basic Course* by Michael Ebner, published by Elektor.

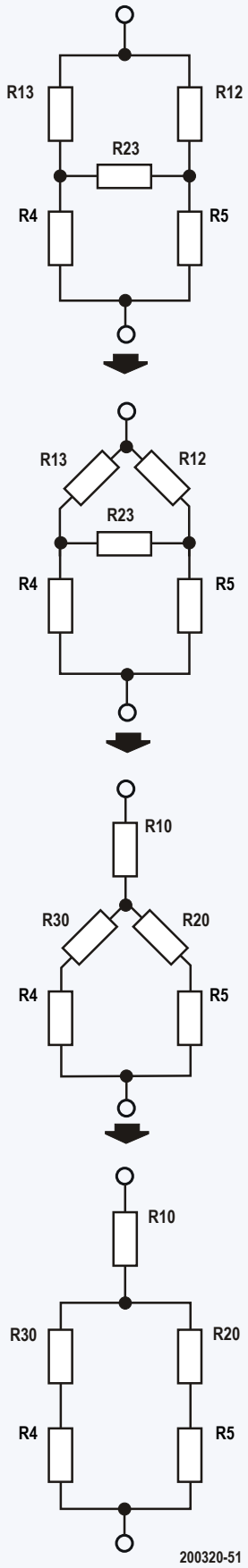


Figure 2: This is how we deal with the H-circuit.

SHOPPING LIST

- **Book: Basic Electronics Course**
  www.elektor.com/13950
- **E-book: Basic Electronics Course**
  www.elektor.com/18232

The Vinyl-Cutter

Returning to the golden-age of LPs

By **Eric Bogers** (Elektor Netherlands)

In today's era of digital audio and high-quality CDs, the analogue vinyl disc has not quite played the Last Post - on the contrary, the number of turntable and gramophone record lovers is on the increase. And, not only that, the true aficionado not only plays LPs but records music onto vinyl themselves!

For faithful Elektor readers, Quentin Therond is no stranger. Among other things he was the winner of the ESP32 contest in 2018 with his design for a 'connected' cocktail mixer [1]. But he also loves the good old vinyl record - so much so that he built the equipment neeed to cut his own records. (That equipment is also for sale ready-made [2] [3], but that is expensive and self-built is, of course, much more fun).

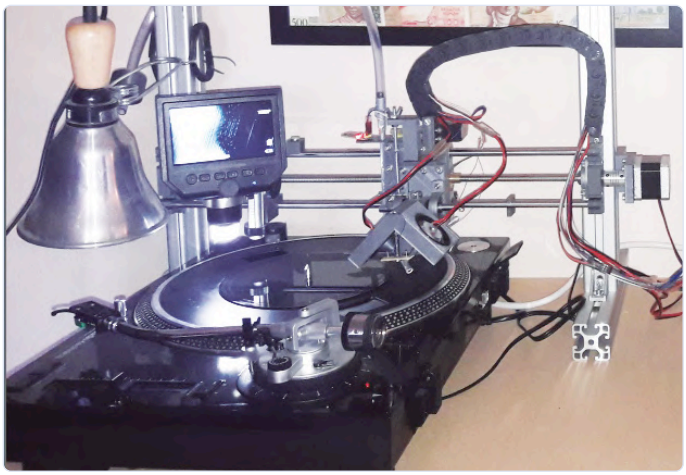


Figure 1: The heart of the setup is formed by a high-quality turntable.

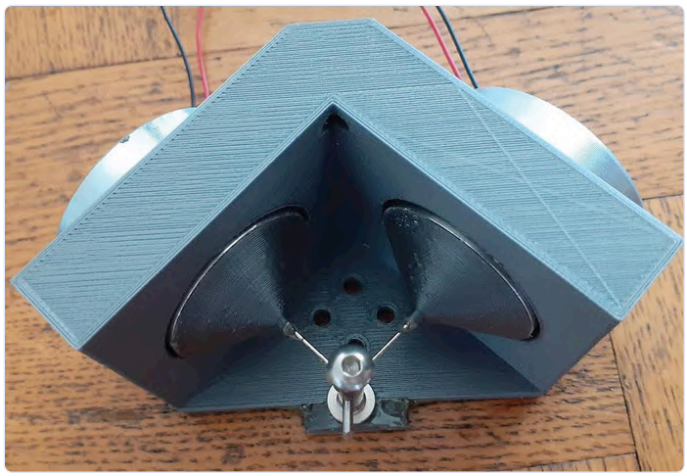


Figure 2: DIY head for the embossing process.

In the film "O Brother Where Art Thou" from the year 2000, a Presto recorder appears in one scene - a wonderful machine (according to enthusiasts) for recording music or speech directly onto a gramophone record. This was Quentin's first introduction to 'direct to disc' recording.
In **Figure 1** you can see the setup Quentin built to cut his own records. We let Quentin provide the description:

"The aim is to engrave the audio signal pattern in the form of a groove into a mould. This is done in real time with the stylus or needle mechanically linked to a loudspeaker or transducer that reproduces the signal to be recorded. The vibrations of the diaphragm are pressed or cut into the mould by the stylus. Thus, the groove should theoretically exactly represent the original signal."
"First of all, the frequency characteristics of the audio signal must be adjusted. During recording, the high frequencies must be amplified and the low frequencies attenuated. During playback, exactly the opposite happens. This is the well-known RIAA characteristic [4] that prevents low frequencies from generating too great a deflection of the stylus, and high frequencies too small a deflection."

Embossing or cutting?

"There are two ways of recording sound on a vinyl record: 'embossing' and 'cutting'. With the first the mould is compressed in order to record the signal; an extremely hard stylus ('stellite') is used for this. With the second method the groove is cut with a sapphire or diamond stylus in a layer of lacquer. This is more expensive but results in the best sound quality."
Figure 2 shows the embossing head (with transducers and stylus) that Quentin built, and **Figure 3** a DIY cutting head with loudspeakers and a diamond needle.

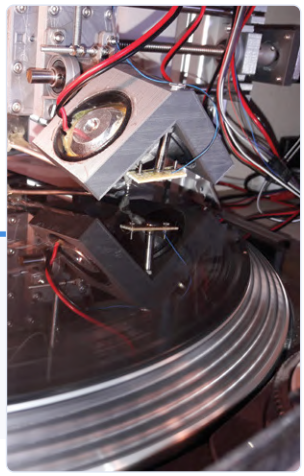
Mono or stereo?

"My cutting head has been built for stereo, with two loudspeakers that are mounted at an angle of 90°. To this, a diamond needle is attached. This way the horizontal displacement of the groove corresponds to the sum of the left and right channels and the variations in depth of the groove corresponds to the difference of both these signals."

How many RPM?

"Depending on the speed of the record (33 or 45 RPM) the cutting head has to be moved faster or slower from the outside to the inside. For each revolution the displacement, for good quality, has to be about 0.1 mm. A 30-cm record at a speed of 33 RPM can record about 20 minutes per side. An extremely accurate slide is required for moving the cutting head with such precision (**Figure 4**). For driving it, a motor with gearbox or a stepping motor is used that generates as little vibration (and sound) as possible. I used a stepping motor myself, but this really vibrates too much, so there is still some room for improvement."

200321-04



WEB LINKS

[1] www.elektormagazine.com/180076-04
[2] https://phonocut.com/
[3] www.elektormagazine.nl/news/zelf-grammofoonplaten-maken
[4] https://en.wikipedia.org/wiki/RIAA_equalization

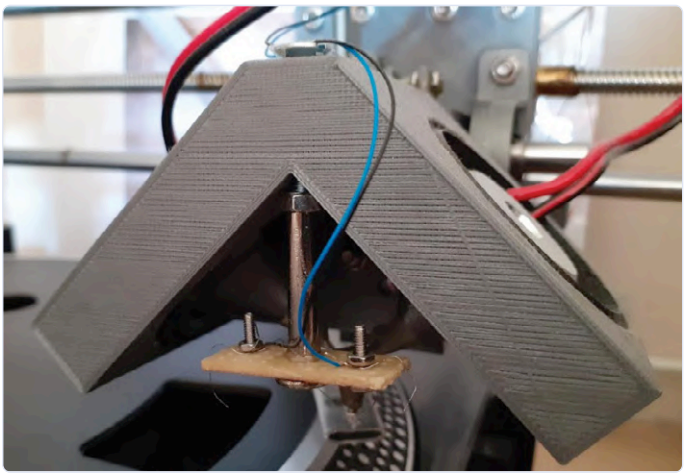


Figure 3: DIY cutting head.

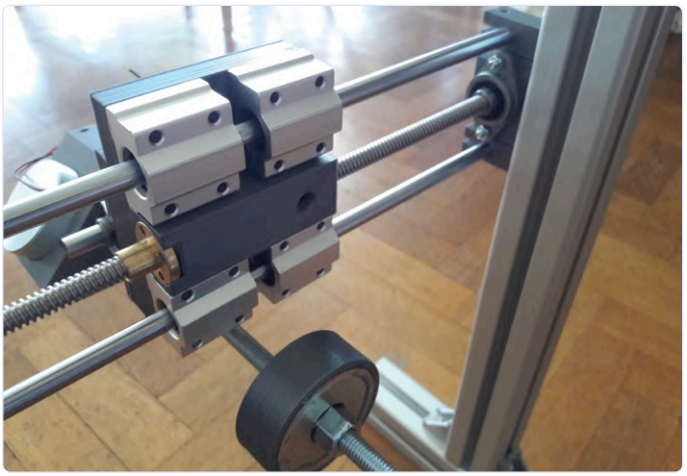


Figure 4: Slider for moving the cutting head.

The 8-Bit Microcontroller and Beyond

Interview with Tam Hanna

By **the Elektor Team**

If you want to build Internet of Things applications, you must understand the 8-bit microcontroller. Tam Hanna, author of *Microcontroller Basics with PIC*, is an excellent resource. He recently shared his thoughts about MCUs and his current electronics projects.

Elektor: Your new book, *Microcontroller Basics with PIC*, is intended to help readers understand and program an 8-bit microcontroller. Tell us why you chose to focus on this topic?
Tam Hanna: Because 8-bit controllers are a fascinating window into the world of microcontroller (MCU) and microprocessor electronics. From a technological point, this is plain and simple — if you understand what happens in an 8-bit MCU, that 400p architecture description of the RISC-V MCU is easier to grok. When I started my engineering career, I had nothing except for enemies. The "Elektronik – gar nicht schwer" series (English: Starting Electronics; Ed.) was tremendously valuable in giving me the ability to learn more quickly, and I am honored to give back, now that I am in a position to do so.
In many ways, the situation facing young engineers today is different. The availability of systems like Python makes getting working systems simple. The Arduino, love it or hate it, insulates you from the underlying hardware. However, in my practical studies, I meet people who run into a wall. Their problem usually is a basic understanding of basic concepts of electronics. A primitive 8-bit microcontroller can be understood completely on an assembly level. This knowledge can then transfer to advanced controller architectures. Given that I still use the PIC a lot commercially, I felt like writing a book is now in order.

Elektor: Who is your target audience for this book? Must a reader be a C programming expert?
Hanna: To be honest, I am not concerned about C programming skills. C has a bad reputation because it is easy to hurt yourself with it. But, like with a Tu-22, problems only crop up if you do stupid or aggressive things. (The TU-22 supersonic bomber was prone to control reversal when flown aggressively.)
I am more concerned about the level of electrical engineering knowledge needed. If you don't at least know the basics of digital electronics, and also understand Ohm's law, you will have problems. Ideally, a reader should have read both the DC and the digital electronics books from the same series.

Elektor: What did you enjoy the most about writing *Microcontroller Basics with PIC*?
Hanna: When I first signed up with SourceForge many moons ago, the site asked me to run through a self-evaluation questionnaire. One of the highest leverage degrees of proficiency was having to write a book about it. In a way, writing this book gave me an ability to sort the knowledge I have collected in my long and sordid career. Plus, of course, the ability of being able to give back a bit of my experience to help others.
Electrical engineering has always been my passion, so working on the text and finding cool examples was an interesting challenge. Being able to help out, especially with a great team, is a great honor.

Elektor: What was the most challenging part of writing the book?
Hanna: Forgive me for sounding cheesy, but this is a personal question and I am going to give a personal answer. The biggest problem I faced was knowing when to stop. While writing this book, I felt the weight of the next generation of cadets on my back. I was thinking of a younger version of myself, sitting in his lab, trying to learn more skills to get that job which lets him break free. There are so many helpful things I could tell him, not just about micro-

controllers, but also about general electrical engineering and test equipment, and only limited time. This is one place where I really have to thank my editor. He did a great job at limiting my work and ensuring that a version of the book would actually become available to the reader so that he can profit from it. And, of course, there always is the hope for a second expanded edition.

Elektor: Intel introduced the MCS-51 four decades ago. Why is the 8-bit microcontroller still relevant?
Hanna: On the risk of sounding like a luddite, there is a strange sense of technology infatuation, especially among novice engineers. When I started my career, I had the pleasure of meeting legendary Sir Bilal Musa from Generali Insurances. He shocked me by telling me that he would never use a product's first generation — whether airplane, whether IT system. This idea has proven itself true in many cases, one example being the Yakovlev Yak-42.
I understand that 32-bit controllers are getting cheaper by the minute. But do you really, always, need the 32-bit performance? Look at what happened when Java went 64bit — in many cases, the longer pointers caused the total system performance to degrade. Especially when working on a low-power system, having to feed 32 memory cells instead of eight cells to store one pointer makes a difference.

Elektor: So, the 8-bit microcontroller is still relevant. Can you now tell us what you like about the PIC family of microcontrollers?
Hanna: One word: simplicity. It is not a coincidence that Massimo Banzi picked the AVR for the Arduino Uno — an 8- bit microcontroller, but optimized for the needs of C compilers. The PIC, on the other hand, was developed from the get-go as a product intended to be programmed by hand. This makes its internal architecture ideally suited as a teaching architecture.
Due to the extremely wide spread of the product, the ecosystem is very mature. Programming devices are dirt cheap. Most PICs are available in DIP housing. And, additionally, as PCs got faster and compiler technology got more advanced, Microchip Technology invested a lot of effort into making the C programming experience almost perfect.

Elektor: Some engineers will say an 8-bit microcontroller is hard to program with C. Do you agree or disagree?
Hanna: Some people say that transporting cargo with a Tu-144 is difficult, and that this makes this supersonic aircraft a universally bad and utterly useless design. No tool can satisfy everybody. If you want to run a complex algorithm, an 8-bit microcontroller probably is not ideal. If you enjoy allocating large amount of memory, you also won't like the experience.
But, let us be honest with one another. For many, if not most jobs, the task at hand is quite simple. In this case, a small C routine is easier to handle than Assembler. If you find yourself banging your head against a wall all the time, switching to a more powerful controller or creating a combinatorial process computer should be a natural choice.
On the other hand, my view, of course, is jaded. Just look at the Palm VII. I actually programmed that thing.

Elektor: Ok, so we've covered the 8-bit microcontroller. Now let's shift to your background and interests. When did you first become interested in electronics? Were you inspired by someone?

The Palm VII portable computer with less than 2 MB of total memory!

Or perhaps you developed your love for electronics by tinkering on your own?
Hanna: In the 1990s and 2000s, people interested in technology were automatically branded Internet-addicted sociopaths. Professionalizing my interest in electronics and IT was a way to escape. Microcontrollers and PDAs don't care much about who operates them. As long as the source code or the circuit configuration is valid, the system works. This merit-based system came as a welcome escape for me, and has kept me happily and profitably at work ever since.
I taught myself a lot of stuff experimenting on my own. This, however, should not disgrace the TGM engineering school in Vienna. Having had access to the workshop was a huge nuisance at the time, but in retrospect, I saw so many things which I would not have seen in another way.
It would be unfair and bad character if I would not thank all the people who have joined me on my engineering journey — whether it was buying apps for PDAs, mobile phone applications, books or my many, many consulting clients. Thank you to all of you. Thanks also to all the people who wrote books, allowing me to learn from them. And, last but not least, thank you for reading this interview and for contemplating to purchase my book. I would be humbled if my text helps you in achieving your goals.

Elektor: Tell us about your current work.
Hanna: OK, this will be a long one. One of the great things in the life of this electrical engineer is that it just never gets boring. There is just so much going on.
The interview is being made in a somewhat funny time. The firm currently is voluntarily quarantined, as are most of our suppliers. This is not so bad though. I recently traded my facility for a large underground lab in Hungary. I'm sitting there, working on all kinds of engineering projects. Before I dive into some specifics, let me point you to my Instagram account (*www.instagram.com/tam.hanna*). Always stay on top of what the Crazy Electronics lab is up to by following my Instagram.
Thanks to stockpiling, my company is able to continue working as normally. The focus is aimed at supplying our clients with engineering consulting work to keep their processes running. In addition to the new products discussed below, I use this opportunity to do some maintenance tasks. I am thinking about recapping my Solartron tabletop multimeter, fixing clogged 3-D printer extruder heads, doing interior renovations, and installing new kitchen fixings. Finally, I might also clean up the laboratory floor so that my wife can finally get around to mopping it.

One oscilloscope for each job.

The next-gen hygrometer is ready for sale.

Elektor: Do you have any engineering tips or advice for Elektor readers?

Hanna: On the risk of making myself enemies and inviting my ex-wife to say that my lab looks like it's located in Havana, Cuba, do not hesitate to buy used things. Having your own vector network analyzer or your own spectrum analyzer is worth its weight in gold. Trust me that your lending place will be closed when you need the kit the most.

In addition to that, I can't stop repeating the old advice to try and break tasks down into smaller problem units. Taking it one step by step, in many cases, is more efficient. Finally, as said in my book, do not be afraid to switch between the hardware and the software domain. And do not hesitate to abuse systems to debug themselves. One really good example for this approach was the debugging of a Kinect application. It was intended to split a picture across threads in a piecemeal fashion. So, thread one handled the left-most part, thread two the middle, and thread three the right-most part. We solved the problem by temporarily making each thread paint each pixel which it touched. After running the modified code, it became clear that the thread picture boundaries were not correct – fixing this, then, was a piece of cake.

Finally, do not hesitate to play with "big machines" and mechanical work. My property was a catastrophe when I took it over, and now is in a more than decent state. The experiences I gathered in mechanical tasks ranging from assembling and adapting furniture to fitting vents on Circo-Geyser radiators directly translate to my electrical engineering work.

Elektor: Engineers and makers understand the importance of learning from their mistakes. What was most instructive engineering-related mistake you've made?

Hanna: This will sound stupid. Normally, I advocate for pushing forward. But my stupidest mistake in recent history was trying to go solo on a TDS754D attenuator repair even though I obviously could not do it due to a tremor at the time. Asking for help or passing off a task to a colleague can, in many cases, save you time and money. Another mistake involves not religiously gaffa-taping hard-to-replace parts of equipment to the carcass during repair. I recently spent a week looking for a small brass cover for a 576 – without it, the fuse for the mains could not be installed.

Finally, always be willing to go where no one has stood before. Using 3D printing in consumer-facing applications was, so far, considered insane. I am pretty confident that owners of Humidors will, soon, appreciate being able to pick the color of their Hygrometer.

Elektor: What's next for you? Do you have a new book, product, or project in the works?

Hanna: Next for me? I'll go and light up a cigar. And while doing that, I will look at one of my upcoming products — the HygroSage. It is the first Humidor hygrometer which does not need calibration at all, has a color screen, and has a 2% accuracy guaranteed over lifetime.

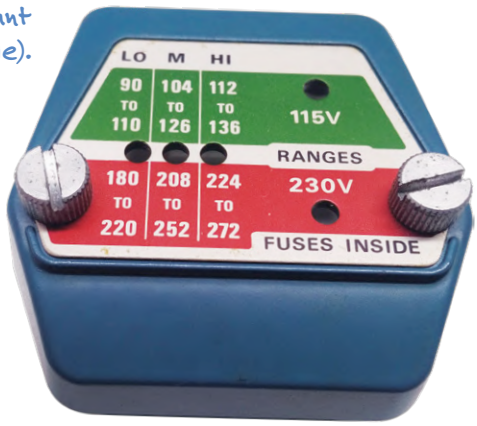
New product number two is the Stinkely series of replacement displays. Danaher makes a pretty penny selling cathode ray tubes for the Tektronix 57x series of curve tracers – we will, soon, throw a wrench into that piece of mechanics. Plus, rescuing these units from the landfill always gives me a warm and fuzzy feeling.

Finally, I work for a US-based fashion start-up which still is in stealth mode for legal reasons. (The USPTO is a strange and fascinating place.) But trust me if I say that we will redefine sunglasses and ashtrays – if you currently circle a $200 (US) ashtray or a $200 (US) set of sunglasses, better wait and let us surprise you.

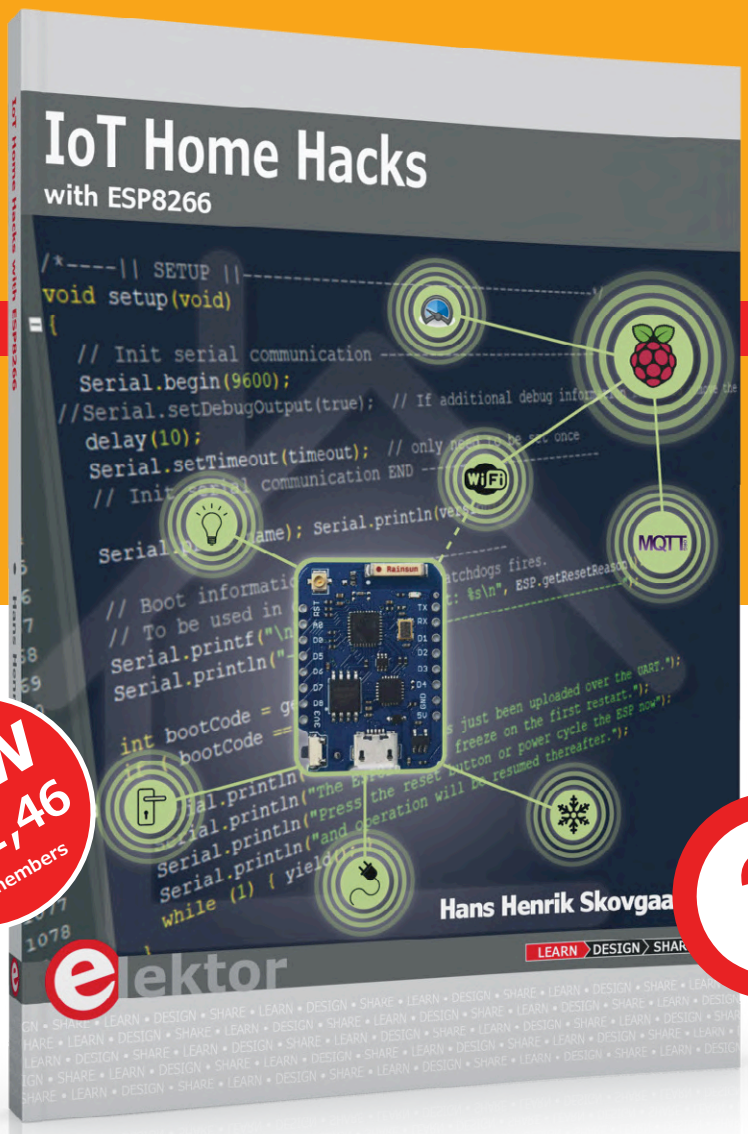
Are you working on a project featuring an 8-bit microcontroller? You can share details and collaborate with other engineers and makers with a free Elektor Labs account (www.elektormagazine.com/labs). Sign up today!

200305-01

Small, but important (and irreplaceable).



IoT Home Hacks with ESP8266



NOW € 31,46 for members

Description:

Ready to start building Internet of Things (IoT) solutions? With a little know-how and the right tools, such as the ESP8266 Wi-Fi module, you can design and build handy IoT solutions for your home or office. In IoT Home Hacks with ESP8266, Hans Henrik Skovgaard introduces the ESP8266 – in the form of the WeMos D1 Mini Pro – and then covers a variety of IoT hacks that you'll find immediately useful.

You'll learn to build the following and much more!

- Colorful smart home accessory
- Refrigerator controller
- 230-V power monitor
- Door lock monitor

The custom software for the IoT devices and PCB layouts are available for free at Elektor.com. Order the book today!

elektor

Read more on:

www.elektor.com/iot-home-hacks-with-esp8266

Small Circuits Revival

From the Elektor suggestions box

Compiled by **Eric Bogers**

For a rainy Sunday afternoon, or for one of those lockdown days when it feels like the walls are closing in on you, small circuits that are easily built on a piece of prototyping board and that invite experimentation are a welcome pastime. Here we present some again.

idea: Petre Petrov (Bulgaria)

Three-way audio amplifier

Sometimes there is a need for a simple 'low-fi' audio amplifier for experiments in the lab or for times when superb audio quality is less important. This is one such design. The circuit presented here may very well belong to the 'low-fi' category but, nevertheless, has a few interesting features as is described in the schematic of **Figure 1**.

This amplifier is intended to be powered using the external power supply from a laptop/notebook. These can be obtained from numerous places for little money and generally supply an output voltage of 19 V_{DC}. A disadvantage of such power supplies is that they often have a lot of noise on their output. That is why an LC filter is used to keep this noise down as much as possible.

For the actual amplifier section, the well-known LM380 (or LM384) was selected with no less than three of them used to build a three-way amplifier. Therefore, there are separate signal paths for the low, mid and high frequencies. The filters that are necessary for this can be bypassed using switches (S3 through S8).

The amplifier also has separate inputs for low, mid and high, but these can be connected together using switches S1 and S2 if required.

The circuit itself is not particularly sensitive and can be built relatively easily on a prototyping board.

A few comments: the amplifier ICs can become quite warm and an appropriate heatsink is absolutely essential. Depending on the impedance of the loudspeakers used, the value of the output capacitor has to be changed as indicated in the schematic.

The LM380, as well as the LM384, have an internally fixed gain of 50 and an input impedance of 150 kΩ. At a power supply voltage of 19 V, the maximum output power (at a THD of 10 %) amounts to about 3 W.

A larger version of the schematic can be downloaded from the project page for this article [1].

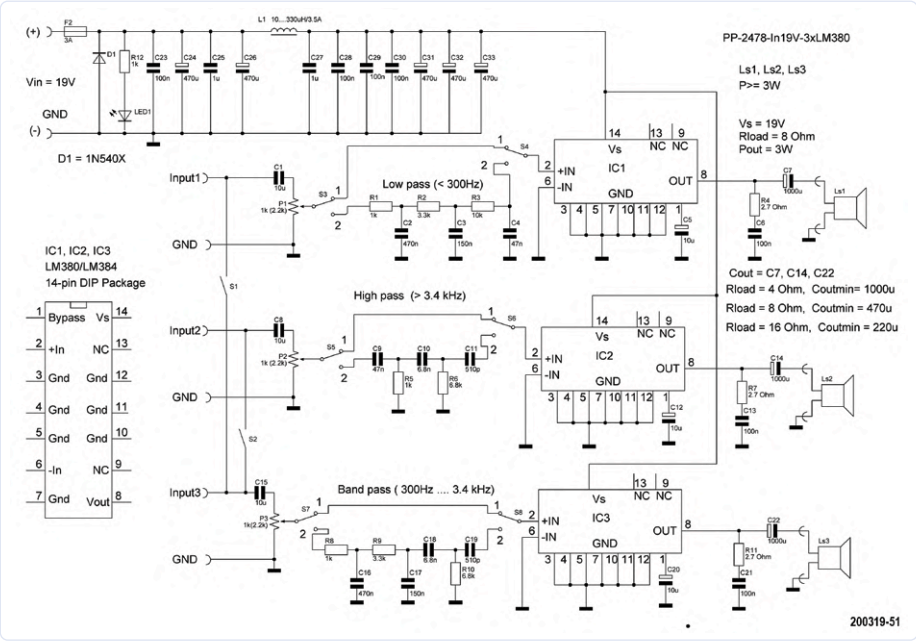


Figure 1: The schematic looks complicated, but looks can be deceiving.

idea: Elex Team

Simple ion meter

That we're presenting such a super-simple circuit here has two reasons. Firstly, in these times of home-office working, the indoor ion climate is very important for your well-being; and, secondly, you'll finally have a use for that antique moving coil meter in your junk box...

It has been asserted for some time that negative ions can have a positive effect on the well-being and mood of people. Admittedly, the positive effect of negative ions is mostly claimed by manufacturers of so-called ionisers; reliable scientific studies have not been able to substantiate these health claims. Nevertheless, in these times of lockdown as a consequence of 'that virus' it is important that the indoor climate is as healthy as possible. And, with the circuit that we present here, you can easily check whether the air around you has sufficient negative ions.

The schematic in **Figure 2** shows that not a lot is required to detect the presence of negative ions. The ions are 'caught' (or collected, if you prefer) with a small metal plate (the shaded square in the schematic). When

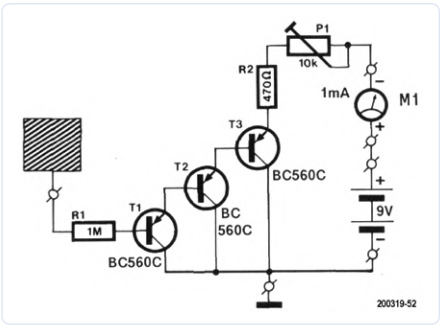


Figure 2: A few components that you are guaranteed to have on hand…

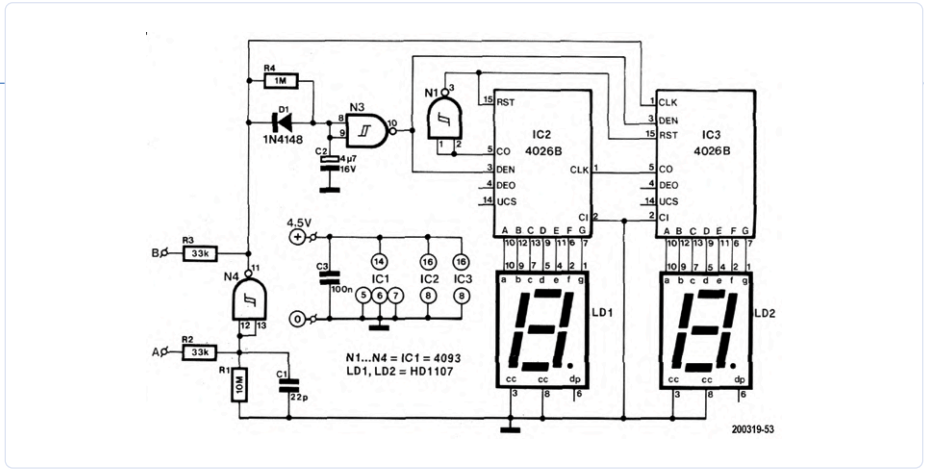


Figure 3: This lottery number generator is easily built on a small piece of prototyping board.

there are positive (and therefore 'harmful') ions, then nothing happens. However, when the 'good' negative ions are in the majority the base of transistor T1 will become negative relative to the emitter and the transistor will start to conduct.

The same is true for T2 and obviously also for T3. These three transistors form a kind of triple-stage super-Darlington. The total gain equals the product of the three individual gains, so there is plenty of gain. It should be clear that even the slightest negative charge on the small metal plate will cause T3 to start conducting. The current through this transistor is made visible by moving coil meter M1. You not only see whether there are negative ions in the environment; the deflection of the meter also provides a rough indication as to how many there are.

Construction on a small piece of prototyping board is perfectly adequate. However, the wire connecting the metal plate and resistor R1 should be kept as short as possible. The adjustment of the circuit with trimpot P1 depends entirely on the local surroundings. A good way to test it is in a bathroom: when you turn on the shower, many negative ions are released. That could incidentally explain why a nice shower is so relaxing…

idea: Elex Team

Lottery number generator

Admit it: you will, at some time, have fantasised about all the things you could do after you win the lottery – buy a house or new car, take a world trip, or tell your annoying boss the truth about their job (and then quickly resign!). But filling in the lottery form (selecting the – hopefully winning – numbers) can be tiring and, (after the draw) can lead to a flaming row. ("I told you to pick 33!" "Yes, but I thought…")

A little electronics can lend a helping hand here. If we assume that all number combinations have the exact same probability then we can use a chance generator and a display to generate actual random numbers (in this case between 0 and 49) and ensure there is never a reason for a quarrel again.

The schematic for the generator is drawn in **Figure 3**. At its heart is an oscillator that is built around N4. Two small metal plates (pieces of circuit board, drawing pins) are connected to points A and B to implement a 'touch button.' As long as the touch button is not touched, the oscillator is stopped and its output (pin 11 of N4) remains high. As a consequence, the voltage across capacitor C2 is also high and the output of N3 is low. This output drives the display-enable inputs of the pair of decade counters IC2 and IC3 and the seven-segment displays remain off.

A brief interlude: the decade counters, type CD4026B, used here have a built-in 7-segment decoder and driver. This allows the common-cathode displays LD1 and LD2 to be connected directly to the outputs of the counters.

Once we touch the button with a finger, the oscillator starts to run at a frequency of a few kHz. At the output is a square wave that is used as the clock signal for IC3. The carry-out output of this IC provides the clock signal for IC2.

When the oscillator is running, C2 is quickly discharged whenever the output of N4 is low. The charging of the capacitor via R4 is much slower. This means that the output of N3 is high whenever the oscillator is running and the displays are therefore also active. Because the counter runs very fast, it will appear that all segments are lit up at the same time.

The instant we let go of the DIY touch button the oscillator stops and the displays show the counter state at that moment. This remains visible for a few seconds until C2 is charged sufficiently again. The displays then turn off and the next number can be generated.

The carry-out output of the counter ICs is high at counter values 0 through 4 and toggles (becomes low) once the counter reaches 5. We use this to ensure the generator does not count past 49. As soon as the count reaches '50', both counters are reset via N1.

The circuit has two minor cosmetic flaws. The first is that it is possible for a count of '00' to be displayed; this is, of course, invalid. And, secondly, the circuit does not check whether the same number is shown more than once. Considering the simplicity of the circuit, it is possible to live with these minor flaws.

And — should you manage to secure the 'big' prize — don't forget about those (us) who enabled your win!

200319-03

WEB LINK

[1] **Project page for this article**
www.elektormagazine.com/200319-03

SHOPPING LIST

› **Engelstalig boek "Electronic circuits for all"**
www.elektor.com/18333

Owon OW18E
Bluetooth Multimeter

By **Harry Baggen** (Netherlands)

At a time when everything has to be 'connected', electronics technicians would like to have a multimeter that transmits their measurement data wirelessly to a phone or tablet. This just what a multimeter equipped with Bluetooth functionality and an accompanying app can do. Here we test the new Owon OW18E Bluetooth Multimeter that offers many features, functionality, and high accuracy for a modest price.

The OW18 series is the most recent series of hand multimeters from Owon. It initially consisted of the OW18A and OW18B, but the OW18D and OW18E were also added some time ago. These offer greater accuracy compared to the A and B versions and a display with a higher resolution (4 ½ compared to 5 ⅚ digit). Just like the B version, the E version in this series is also equipped with Bluetooth.

OW18E hardware

The OW18E Bluetooth Multimeter looks the same as all meters in this series. The fairly large housing feels sturdy, and it is equipped with a blue (removable) protective sleeve made of soft plastic. The rotary switch works quite smoothly, while the large display shows the value in four digits with suppressed 1, so the maximum reading is 19999. Surrounding it are various indicators that are visible depending on the setting. The display can be read quite well, but when viewed from above the contrast quickly decreases. At the back, there is a folding support and the battery compartment for

a 9 V battery. There are four push buttons that enable switching to the second function of any switch position, manually setting of the range, controlling display illumination, Bluetooth control, hold, and relative and duty cycle selection.
The OW18E Bluetooth Multimeter comes with a set of test leads and associated crocodile clips, a cable with a (K-type) temperature sensor, battery, manual, specification sheet, a card with instructions for downloading the software and, finally, a screwdriver for opening the battery compartment or housing.

Measurement capabilities

The number of measurement options is quite large. The most commonly used ranges are of course V and A with AC voltages using True-RMS measurement. There is an extra sensitive V-range of 20/200 mV and an extra sensitive A-range of 200 µA. It is also striking that up to 20 A can be measured (albeit for a maximum of 10 s!), something that is often limited to 10 A for other meters. Of course, the OW18E Bluetooth Multimeter also has an Ohm/diode/continuity measurement mode. Capacitance measurement is possible up to 20,000 µF. The frequency measurement runs up to 20 MHz, which is quite a large range for this price range! The duty cycle can also be measured in this position. The temperature measurement with the supplied sensor goes up to 400 °C. Finally, there is an NCV mode with which a contactless measurement for mains voltage on a plug, line or socket can be made.

Above the display sits an LED indicator that starts flashing when contactless measurements of the mains voltage are detected. This indicator also lights up when operating most functions. In addition to the NCV sensor on the front, there is also a white LED that serves as a flashlight. The meter is (of course) equipped with an auto-power-off (APO) function that switches the meter off after 30 minutes of non-use.

Measurement results

First a note about safety. According to the manufacturer, the meter meets the measurement category CAT III 1000 V/CAT IV 600 V. That is quite a good insulation level and more than good enough for measurements in and around the house.
The basic accuracy of the OW18E Bluetooth Multimeter is specified at 0.1% + 2 digit for the 2 V range and above. When compared to a meter with 10 times the accuracy, the Owon was terribly close with a deviation of less than 2 digits. The results were comparable for DC measurements.
In AC measurements, the deviation was a few millivolts at a measurement value of almost 2 V. This is also fine as the specs indicate 0.5% + 10 digits. A frequency range of 40 to 1000 Hz is specified for AC measurements. In the test measurements, the reading remained fairly accurate up to approximately 1.5 kHz.
The resistance measurement was again much more accurate than the promised 0.3%; above 1 kΩ I came to a deviation of 0.1 - 0.2%. Also, during the capacitance measurement, the measured values deviated only a few digits from the reference meter. These are all excellent results for a meter in this price class.

In daily use

So how does this meter stack-up in daily use? The display is clear with large numbers and is easy to read at almost all angles. The autoranging function is not very fast, but it does works reliably. An interesting detail on the display is the range indication provided below the decimal point in very small numbers. The functions of the various pushbuttons are quickly understood and, when you operate the built-in beeper, the indicator LED lights up. The pass-through beeper works very nicely when you need to solely rely on it. The capacitance meter works well, even for large values. For a 2200 µF capacitor, the meter took approximately 8 s to determine the capacitance.
The lighting of the display provides for a legible output in a dark environment and, at the same time, the flashlight LED on the top lights up. Unfortunately they cannot be operated separately, but this was no issue for me. I was less impressed by the NCV function of the OW18E and it is quite insensitive. Sometimes you can sit on top of a mains plug with the sensor and the detector does not engage, so I wouldn't trust it entirely.

Everything delivered with the OW18E.

The NCV sensor is located on the top along with an LED.

Bluetooth

In addition to the high accuracy for this price range, the Bluetooth function is the main feature of the OW18E Bluetooth Multimeter.

The 9 V battery is in a separate holder that clicks and screws onto the back.

The meter has a BLE4.0 module that should ensure a stable connection and low energy consumption. I activated the Bluetooth function on the meter for several hours without the battery giving up. The accompanying MultimeterBLE app is available for Android and iOS users.

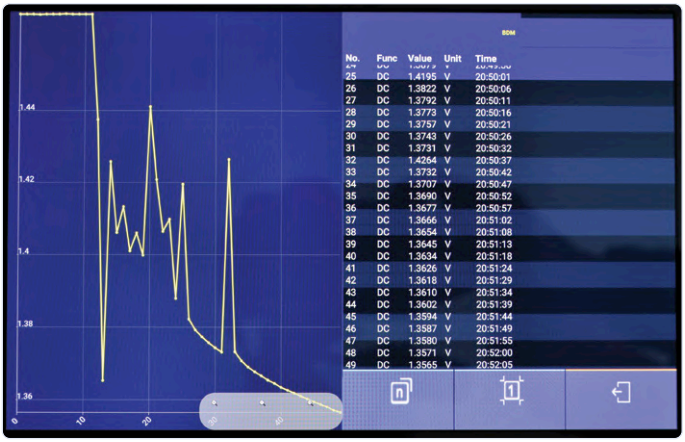
After installing and starting the app, turn on the Bluetooth function on the meter and it will appear in the app as "BDM". Once selected, the app will connect and the reading will appear on the display. You can operate the functions of the push buttons from the app. However, the function of the rotary switch must still be done manually. The app also offers a data logger function with a number of setting options, such as the logging interval. The logged values can be stored and sent in a file. You can also log the meter independently from the app, after which the Bluetooth connection can be disconnected. The measured values can be retrieved later via the app. The app can also connect to two meters at the same time.
It is a little unfortunate that the app only works in landscape mode and by default shows a screen for two meters. Most users will only have one meter at their disposal. However, the app works smoothly overall, the connection is stable, and the data is quickly transferred from the meter to the app. This wireless connection is very useful if you cannot or do not want to sit next to your meter continuously, and the information in the app is no less than the information on the meter's display. This meter therefore offers a good alternative to the popular, but no longer available, Mooshimeter.

A nice touch: below the decimal point the selected range is displayed.

Great price, high accuracy

For less than €70 you can make an excellent investment with the Owon OW18E Bluetooth Multimeter. For your money you get a fairly sturdy device that not only offers a lot of measurement options but is also surprisingly accurate. Added to this is the Bluetooth functionality that makes it possible to monitor measurements and log measurement data remotely. What more do you want for that money?

200322-04



Logging data in the associated app.

SHOPPING LIST

› **Owon OW18E Bluetooth Multimeter**
www.elektor.com/owon-ow18e-bluetooth-multimeter-20000-counts

Interactive

Corrections & Updates || Questions & Answers

Compiled by **Clemens Valens** (Elektor Labs)

Updates of and additions to projects published in *ElektorLabs* magazine, spiced up with tips & tricks, tech advice, and answers to readers' questions.

New firmware for old project

My interest was drawn by the article "Universal PWM Driver" published in the July/August 2010 edition of your magazine, because it uses a rotary encoder to select options in a menu. I tried several times to program a PIC16F628A [as used in the project, *Ed.*], but the software refused to work. The software was, I think, originally developed in C/C++ with probably a CCS compiler.

Owning an EasyPICv7 board and a mikroC license, I ported the original software only to discover that it still didn't work. I therefore analyzed the source code in detail, modified a few lines of it, and programmed a PIC16F88 with the resulting HEX file (the PIC16F628A originally used lacked RAM, the new LCD library probably consumes more resources than the one used by the developer Alexander Ziemek). This way I managed to get a fully functional system, although certainly perfectible.

I also added many comments throughout the code, making it easier to read and understand. It goes without saying that I worked on this project as an amateur; I am absolutely not a professional software developer.

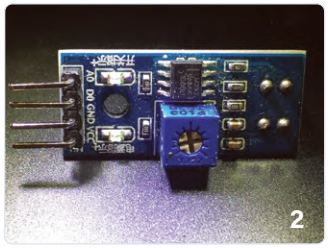
I hereby offer you my work [1] as a ZIP archive together with the schematic that I have slightly modified, because I do not use the R/W pin of the LCD, and which is now connected to ground. This frees the RA2 pin of the processor. The spirit as well as the functions and features of the original design remain unchanged.

Philippe Le Guen

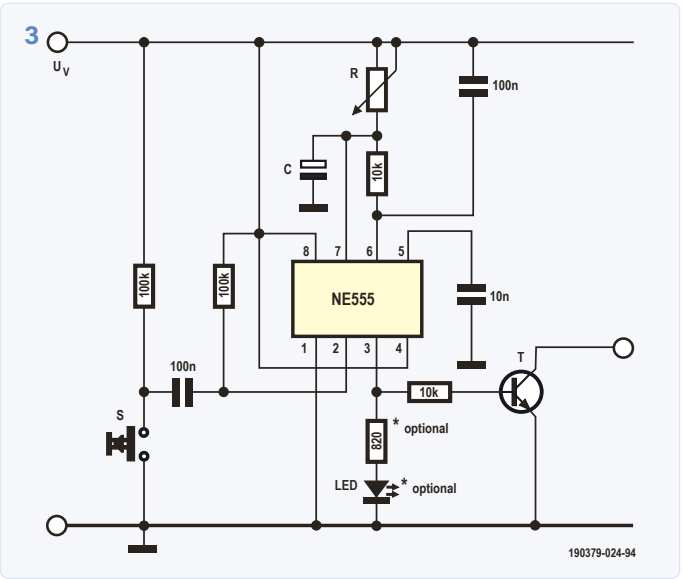
www.elektormagazine.com/magazine/190379-D-01

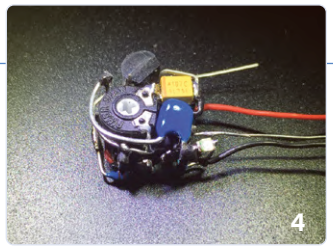
Improved sensor for Water Flow Monitor

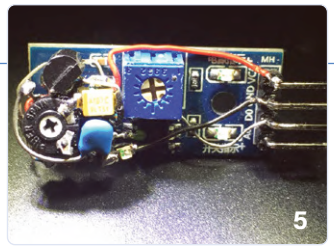
After reading the article "Waterflow Monitor with ESP32" by Denis Lafourcade in the July & August 2019 edition of Elektor, I immediately realized that such a system would be very useful to me. But the first step proved to be difficult: I could not find a suitable sensor for my water meter. An inquiry to the manufacturer remained unanswered and so I had to rely on a DIY solution.

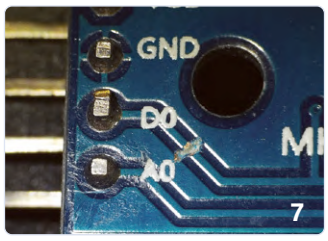


A first test with a Hall sensor was unsuccessful as the water meter contains no magnetic parts (to prevent fraudulent acts, as I found out later). A semicircular shiny surface on the litre pointer seemed to offer a possible solution (**Figure 1**). Maybe it could be used to detect the rotations of the pointer? Since I had ordered some reflective infrared optical sensors some time ago (**Figure 2**), I looked for mounting possibilities for one of these on the water meter. From Thingiverse [1] I downloaded 3D-model files for a suitable sensor housing, printed one and put a sensor in it. This assembly was attached to the water meter with suction cups. The rest of the water consumption monitor was built quickly too.









The first test gave disappointing results: the monitor's readings were far too high. A look at the Arduino sketch and subsequent analysis of the sensor's output signal revealed the reason. The sketch assumed 500 ms pulses, but my infrared sensor gave different values. Not only depended the length of the pulses on the water flow rate, it also happened quite often that the shiny segment stopped right in front of the sensor, resulting in a permanent high level.

What to do?

Immediately I thought of the good, old timer IC NE555, but a microcontroller-based approach would have been possible too. Since I wanted to keep it all analogue and as I happened to have a few 555s lying around, I opted for this solution. I used the 555 in a nonretriggerable monostable multivibrator (MMV) configuration (**Figure 3**). This MMV can only be triggered when it is idle. If this happens, it will produce a pulse of about 500 ms at its output. This method also avoids false triggering when the circuit is switched on.

Construction of the circuit was not easy as all the parts, including the 100 µF electrolytic capacitor, had to fit in the limited space left in the sensor housing. To achieve this, I used some parts in SMT packages (**Figures 4, 5, 6**).

To connect the circuit to the sensor, you must cut the track leading from the IC LM393 on the sensor board to the output on connector Do (**Figure 7**). Pin 7 of the LM393 is then soldered to the input of the MMV; the output of the MMV is connected to the connector (Do).

On my request the author has also added brief instructions to the project page at Elektor Labs [2] for setting up the IFTTT service.

Hans Schneider

www.thingiverse.com/thing:2749407

www.elektormagazine.com/labs/waterflow-monitor

Build a Wearable Vibration Monitor

Long-term exposure of your body, or parts of it, to vibrations may result in injury. But how do you measure the risk? This project might help to find out. Elektor Labs contributor 'sixbacon' is a physicist and engineer with an interest in health and safety who wanted to get a feel for the levels of vibration which are harmful. He also wanted to assess DIY tools which are essentially unregulated. Since he also likes building things, he saw making a monitor as an interesting challenge.

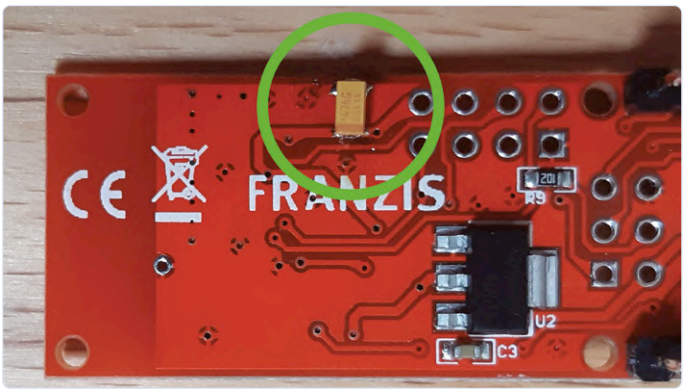
The vibration monitor consists of a mount carved from wood (but a 3D-printable design can easily be developed). The shape is a simple 'H' to allow it to fit between the fingers of a gloved hand and place the sensor against the tool. An ESP32 module with OLED display

and a LIS331 accelerometer were fixed on top to allow the operator to read the display. A small USB battery pack powers the unit and can be tucked into the operator's sleeve while in use.

Although originally designed for measuring vibrations in the user's hand and arm caused by handheld power tools like drills and grinders, there are probably many other fields of application for it. All you need to do is place the sensor in the right spot.

www.elektor-labs.com/1879

Improving the Pretzel Board



A while ago when playing around with the Pretzel Board [1] I experienced some "teething troubles". The board didn't do what I expected from it, and after some searching online I found out that the power supply of the WLAN part (even on current boards) can be too unstable.

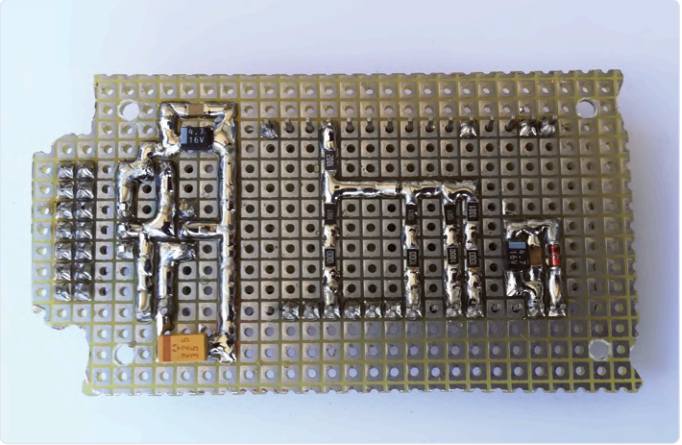
The suggested remedy was to add a big 100 µF buffer capacitor on the board's connector to improve the stability of the WLAN

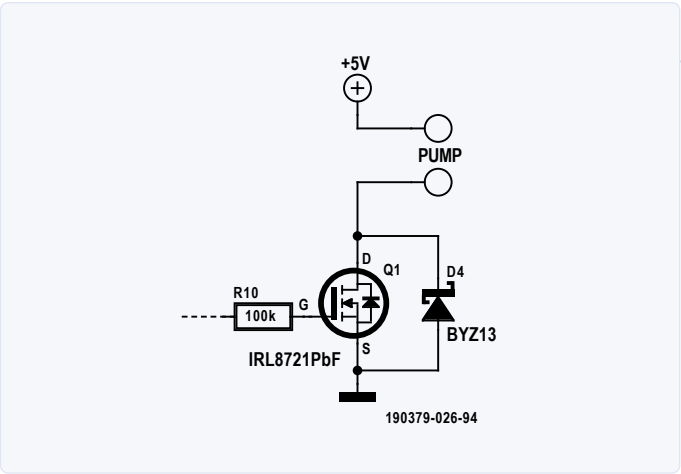
part. This does indeed solve the problem, but I preferred a more elegant mechanical method than blocking the connector with a big, fat capacitor.

You can see the result in the photograph. The electrolytic capacitor is a SMD type with a value of "just" 47 µF, which is "inconspicuously" attached to the bottom of the board. To do so, I carefully scraped away some solder stop mask from two tracks at the correct place and soldered the capacitor there (with a suitable soldering tip). A 100 µF-type might have fit as well, but I didn't have one.

Uwe Werner

www.elektor.com/pretzel-board-iot-wifi-board





That was exactly what the designer thought too. And with a pump from SOS-solutions and a few additional parts it is not difficult to achieve (see the schematic). The pump is controlled by a MOSFET (IRLB8721PbF), which in turn is connected to the output of the circuit via a 100 kΩ resistor (instead of the buzzer). A Zener diode protects the transistor.

When using the pump, the duration of the pulse produced by the one-time shot must be increased from 100 ms to one second by making the value of C4 ten times bigger (i.e. 100 nF); in addition, the period of the astable multivibrator must be increased to e.g. three seconds by using a value of 2.7 µF or 3.3 µF for C1.

With this modification and a suitable water reservoir, the plant can survive unguarded for months.

www.sossolutions.nl/1150-peristaltic-liquid-pump-with-silicone-tubing

How to hand-solder surface-mount devices (SMDs)

Today it has become almost impossible to build electronics projects without running into those tiny surface mount parts better known as SMDs. Even though reputedly hard to solder, many people have come up with ways of using them for prototyping anyway. Here is one trick suggested to us by Antoni Gendrau. If you have another, feel free to add it as a comment to the project and let the world know about it.

https://www.elektor-labs.com/1913

Stargate NeoPixel watch

Elektor 07/2020, p. 76 (200208)

In the description of Peter Neufeld's home laboratory and the Stargate NeoPixel watch that he built there (July / August 2020), we wrote that the wooden insides represented many hours of skilled fretwork. That was confusing, as Mr Neufeld told us: The interior is in fact a kit from the company UGears – all you have to do is press the laser-cut parts out of the wooden plate and assemble them (without glue!).

https://ugearsmodels.com/de/date-navigator.html

Upgrade for Dry Soil Alarm

Elektor 07/2020, p. 39 (200213)

In the July / August 2020 edition of *Elektor* we described a *Dry Soil Alarm*, a small circuit that warns acoustically and optically if the soil of a plant is too dry. Useful, but it would be even more useful if the plant were watered automatically.

Simpler Inrush Current Limiter

The designer of this handy little circuit in the March/April 2020 edition is Kurt Kühni from Switzerland and not someone from Elektor Labs as was suggested (even though we would have liked to come up with it ourselves). Please accept our sincere apologies for any confusion caused.

Steeped in Electronics

Lead-free soldering and EU regulatory zeal

By **Ilse Joostens** (Belgium)

When I was very young, in the early 1970s, we lived in a row house with lead water pipes. When I was five years old my family moved into a brand-new house without lead pipes, but with flooring that contained asbestos and had been treated with pentachlorophenol. I must say that the latter sounds really ghastly, and with all the bad news now about things we used to like so much, it seems like a miracle that I am still alive and able to write this column. The world has clearly become a lot more dangerous since I was young, and with the introduction of the RoHS regulations in 2006 the European Union in particular apparently found it necessary to launch a major crusade against the use of a number of hazardous substances in consumer electronics. For some mysterious reason, lead is regarded as the main culprit, and now it is even supposed to be banned from ammunition and fishing weights.

Lead-free baking

If you do soldering, make repairs or build prototypes as a hobby, you need not read any further, and you should get down on your knees and thank the gods that you are allowed to use lead-based solder. If like me you put together commercial products, you are out of luck and condemned to working with lead-free solder. Strictly speaking, you could still use lead-based solder for your prototypes, but to avoid the risk of contamination I use only lead-free solder.

The period shortly after the first Restriction of Hazardous Substances (RoHS) regulation took effect in 2006 was fairly dramatic for the electronics industry. Some lead-free processes were not yet ready for use, and there was only limited experience with the new methods. During that period I repeatedly woke up drenched with sweat after yet another nightmare about razor-sharp tin whiskers constantly growing between the leads of fine-pitch SMT components, ultimately resulting in smoke signals.

As far as automated soldering is concerned, most of the problems have been solved by now, and in practice the tin whiskers and other potential disasters have turned out to be minor issues, although I have the impression that modern consumer electronics fails faster and more often than in the past. Of course, this could also be due to the quality of the components concerned. With the growing number of low-cost imports from Asia, it has systematically declined.

Unfortunately, manual lead-free soldering is still a bit of a challenge. And if we can believe the countless photos circulating on the Internet, soldering on computer motherboards is very hard on the fingers, and the higher melting temperature of lead-free solder certainly doesn't help. To see this for yourself, just Google for pictures matching the search term "solder fail", and be sure to have your burn ointment close to hand. With my old soldering stations, soldering relatively large through-hole components on PCBs with plated-through holes was not exactly easy with lead-free solder, especially on solder pads in or next to large copper planes. The irons were usually set to a temperature of 420 to 450 degrees C, and in some cases I needed two irons at the same time to get the solder to melt. It's hardly surprising that I went through solder tips pretty fast.

Although working with lead-free solder is a good deal easier using modern soldering stations with interchangeable active tips, my feeling is that soldering on large copper surfaces is still not straightforward. Desoldering through-hole components and SMD rework are still tricky tasks, leading to more failed components and half-finished products. In the very rare cases where I work with lead-based solder, it's always a relief.

If you intend to get into electronics or you occasionally solder something as part of your hobby, I can only recommend that you avoid lead-free solder. Good soldering stations with active tips are expensive, and actually there are no specific health hazards from working with lead-based solder. Of course, you shouldn't eat it. Lead-free solders usually contain a somewhat more aggressive flux, and the flux vapours are not especially healthy. This doesn't mean it's okay to start smoking lead-based solder like a chimney – that should be obvious.

Recycling

It's also questionable whether banning lead from electronics is really that much better for the environment. Instead of the current cheap disposable electronics, it would have been better to aim for higher quality and more durable products. This brings us to the recycling of products, and there as well the government busybodies and bureaucrats know how to make your life miserable. In most European countries there are organisations for recycling discarded devices and reusable waste. In Belgium we have Recupel, Bebat and Fost Plus, among others. I am fully in favour of recycling, but association with the above-mentioned organisations is not only expensive but also involves a lot of paperwork. Fortunately, I'm not bothered much with this as long as I do not sell completely finished products or batteries and keep the use of plastic packaging within reasonable limits. From what I hear from colleagues and competitors in Germany, the situation in Belgium is not so bad. Stiftung EAR, the German equivalent of Recupel, takes its role very seriously and has already imposed hefty fines in some cases. In Germany, rules are rules.

Despite all the regulations and prohibitions, components such as mercury switches and light-dependent resistors with cadmium sulphide (shiver!) are still readily available from Asian sources, not to mention some potentially lethal products that do not comply with any standards but are still for sale in shops here. Is this a sign of a double standard amongst our European policy makers?

200082-02

Promising Project:

New 50 Hz – 2 MHz LCR meter

Accuracy and ease of use

By **Jean-Jacques Aubry** (France)

A little over seven years ago, Elektor published a series of articles on my 0.05% LCR Meter. More than 500 readers constructed one from the kit of parts offered by Elektor. While I was happy with this success, it also motivated me to design a new LCR Meter that focused on ease of use rather than extreme precision.

Elektor Kickstarter?

This project is one of the first to use the new Elektor Kickstarter module of our online **Elektor Labs**. Our goal is to use this approach to attain insight into the interest for this project from our community, as well as to determine a suitable number of kits to produce.

How it works:

1. Look up the project online at Elektor Labs (www.elektor.com/lcr)
2. Click 'Read more' in the right column, next to the rocket.
3. We will provide (limited) price and product details as available at the time.
4. If you are interested in buying at a later time, click 'Back This Project.'
5. Enter your email address.
6. Done.

There is no need to pay upfront and there are no strings attached. You can also cancel your interest at any time. We will inform you via email about the kickstarter status, and the production and release date.

1 more thing:

All our backers receive a nice discount when the product arrives in our store!

Better name?

Elektor Kickstarter: obviously we cannot continue to use the kickstarter name. We currently have a contest asking you to help us find a better name at www.elektor.com/kickstarter - why not join in?

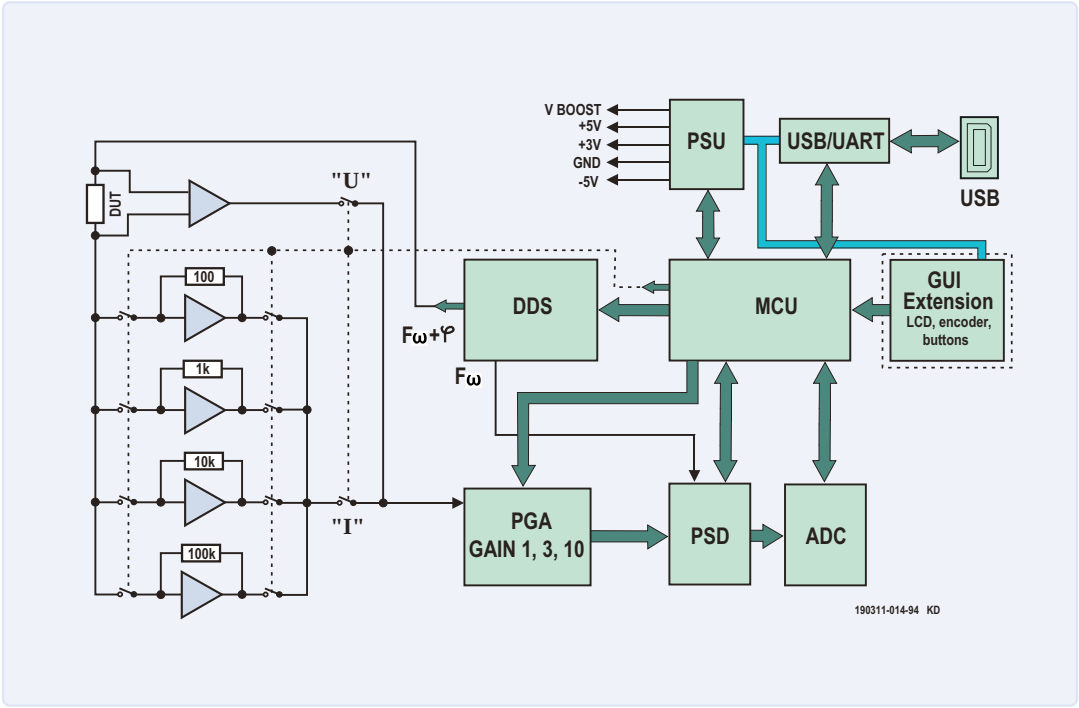


Figure 1: Block diagram of the LCR meter. All the core functionality is on the main board with the display, controls, and possible extensions on a second board (GUI Extension board shown as a dotted line)

The challenge I set myself comprised ease of measurement with extended functionality:

- Test frequency ranges from 50 Hz to 2 MHz
- Choice of 4 test voltages: 100 mV, 200 mV, 500 mV or 1 V rms.
- Addition of DC bias voltage of up to 5 V for capacitors and DC current of up to 50 mA for inductors.

To overcome any reticence that a project of this class may (wrongly!) inspire, I paid particular attention to the ease of setting it up, such as the inclusion of automated calibration. I also focused on its user interface by implementing a rotary encoder and five associated push buttons with multiple functions; and a 240 × 128 pixel graphic LCD display for easy navigation of the menus, frequency configuration and, of course, display of the measurement results.

Pushing the limits

Impedance (Z) is an important parameter to measure in passive electronic components (resistors, capacitors, inductors). To determine the impedance, at least two values (amplitude and phase) are required, usually the voltage at the terminals of the component and the current through it. Like its predecessor, the new LCR meter uses the self-balancing bridge method with a simple operational amplifier for current/voltage conversion.

This method is simple, its precision is good, and the cost is reasonable. The main disadvantage is the limitation of the operational amplifier over the frequency range defined. Most similar testers have an upper frequency limit of 100 or 200 kHz. I have succeeded in pushing the high frequency limit to **2 MHz** without a huge cost increase or compromising the simplicity of construction. The project will be described in detail in the next issue.

The completed design comprises two boards. The first is a main board whose functions are shown in the block diagram (**Figure 1**) and comprise of:

- **Input circuit**: The device under test (DUT) has a test signal applied to it. To reduce the influence of the test leads, the measurement is made with a 5-connection configuration.
- **Sine wave generator:** A Direct Digital Synthesis (DDS) block generates the test frequency. This simultaneously generates any frequency in the range of 50 Hz to 2 MHz together with a signal of the same frequency, but variable phase, for the synchronous detector.
- **Programmable Gain Amplifier (PGA):** Provides a gain of 1, 3, or 10 to supply the phase detector with an optimal input signal.
- **Phase Detector (PSD):** Requires a perfect square wave at the same frequency as the generator but with a relative phase difference that may be varied. This phase variation of the switching signal allows measurements of the component in phase or in quadrature of the input signal.
- A **microcontroller**, 24-bit **ADC**, **USB bridge** and **power supply**.

The second is a display and user interface board, integrating:

- Graphical LCD display.
- 5 push buttons with multiple functions (shown on the screen) and a rotary encoder.

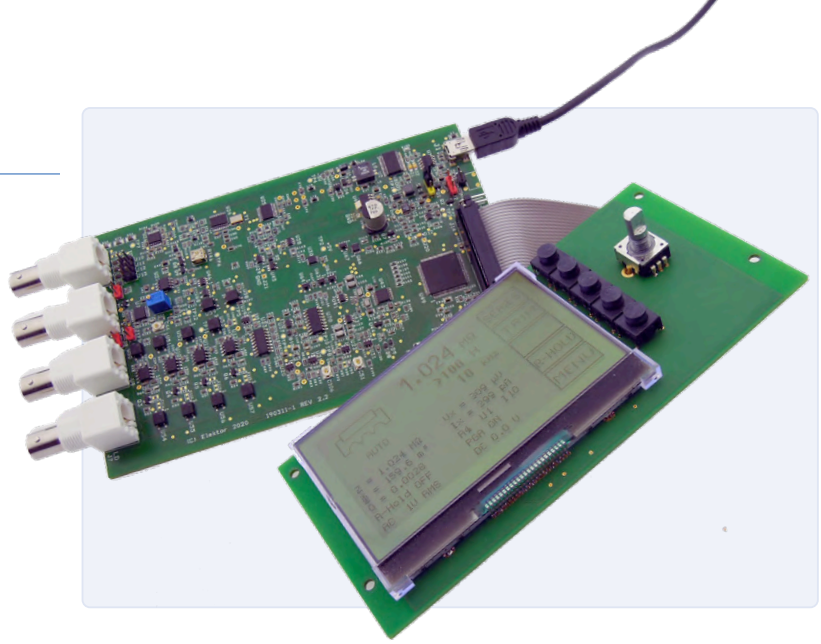


Figure 2: The combination of main and extension board provides a self-contained device. The main board can be connected to a PC.

Only with your support

The specifications of the LCR meter are summarised in the below table.

I would like to request that you follow my project on the Elektor Labs site at www.elektor.com/lcr. Before we start production of a kit, Elektor has launched a support campaign to gauge interest. If you would like to get your hands on a kit you can register, without making any formal commitment, online. The production of the kit will be started as soon as we have registered 150 confirmations of interest. In exchange for your early commitment you will be offered the kit at a reduced price.

The LCR meter kit will include:

- **Main board** preassembled with all SMD components soldered.
- **Display board** preassembled with all SMD components soldered.
- **Through-hole components** for both printed circuit boards (backlit LCD display, connectors, push buttons, rotary encoder, button).
- **Ribbon cable** to connect the main board and display board.
- **Mini-USB cable** for connection to PC and software updates.
- **Aluminium Hammond case** with drilled and milled panels.
- **Screws**.
- **Kelvin clip** with test cable with four BNC plugs.
- **User Manual**.

200309-03

Main Technical Specifications

Display	• Parameter values (principal and secondary) • Equivalent circuit: series or parallel • Frequency • |Z| • Φ • Q or D • Voltage and current of DUT (V_x and I_x) • Test voltage (AC) and polarisation (DC) • Range hold status (R_Hold) • Labels for multi-function buttons (right-hand side)	
Measurement range	**Parameter**	**Value**
	L	10 nH ➞ 100 H
	C	1 pF ➞ 100 mF
	R, |Z|	10 mΩ ➞ 100 MΩ
	Q	0 – 5000 for display
	Φ	– 90,00 ° / +90,00 °
Test frequencies	50 Hz to 2 MHz in 54 predefined steps and any frequency within the range	
Power consumption	With display/control board	5 V / 420 mA
PC software	for Windows and macOS	

Test conditions:

Open-circuit test voltage	4 values : 0,1 V_{rms}, 0,2 V_{rms}, 0,5 V_{rms}, 1 V_{rms} ± 5 %
DC Polarisation	For C: 0 to 5 V | For L: 0 to 50 mA

Accuracy of main parameter (R, L, C) :

Conditions*	After calibration and use of a frequency test in accordance with the DUT (at the high frequencies the parasitic components take a lot of importance)
Accuracy	Up to ± 0.1% ±1 digit

Various :

Test connections	4 wire Kelvin via BNC connectors
Power suppy	5 V_{DC} ± 5 %, from mini-USB connector

Error Analysis

Tips on FMEA, High Current and More

Compiled by **C.J. Abate**

Excessive electromagnetic interference (EMI), a fried circuit board, an exploded battery, and faulty sensor readings: these all can result from a simple engineering error. To improve as a professional engineer or serious electronics maker, you must learn from your engineering errors, as well as the mistakes of others. In a new series called "Error Analysis", we share insights from Elektor community members about their engineering-related errors and what they learned from the experiences. And, of course, we highlight tips and tricks that will help you enhance your design, testing, and programming skills.

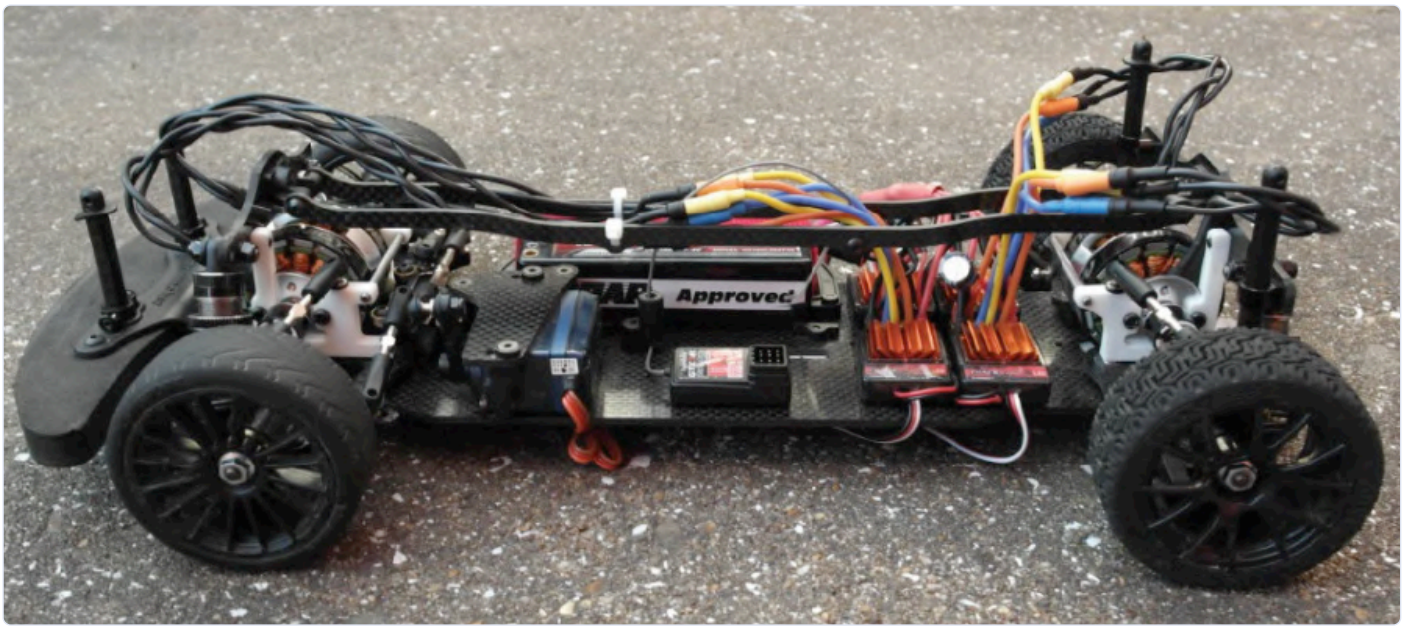


Figure 1. Rajesh Nakarja's RC project.

Improve Design Reliability with FMEA

No matter how complicated your engineering project, applying Failure Mode and Effects Analysis (FMEA) can mean the difference between success and disaster. Stockholm Sweden-based engineer Rajesh Nakarja knows this well. Here he shares the story of a problematic Arduino-based RC design and, of course, what he learned from the experience:

"As a student, I loved RC modelling. My degree at the time, in embedded control, led me to develop a custom DSP controller for torque vectoring on 1/10 scale RC cars. The vehicle, which featured four independently driven wheels and various sensors, was capable of reaching extreme speeds normally reserved for competitive racing on a track (**Figure 1**). With all the real-time processing, however, the vehicle was able to maintain stability at full power by being able to detect loss of traction, as well as self-correcting itself with gyros.

The code itself was fully deployed from Simulink coder onto a custom STM32F4 board and featured numerous code safety features. Over/under flows, loss of radio or sensor data, all such things would resolve safely and bring the vehicle to a stop. Lots of testing and simulations showed this to work well, and I was confident that it would be safe to actually drive.

> Modern tools and workflows, such as model-based design, allow for automated testing and coverage during development.

While it was only a couple kilograms, the speed and power of four 40A BLDC motors meant that crashes could still be quite devastating to anyone or anything it may hit. As a result, I was quite careful about using it outside and was always wary about running it around people or traffic.

To control the car, I had hacked together an Xbox controller to an Arduino-ZigBee bridge, which provided real-time steering and throttle control to the car. While the car controller itself was engineered to be reliable and robust, the Arduino controller was more of a hack, and the only safety check added was to shut down the car in case of loss of radio. At the time, that alone seemed like a reasonable enough fallback for a simple controller.

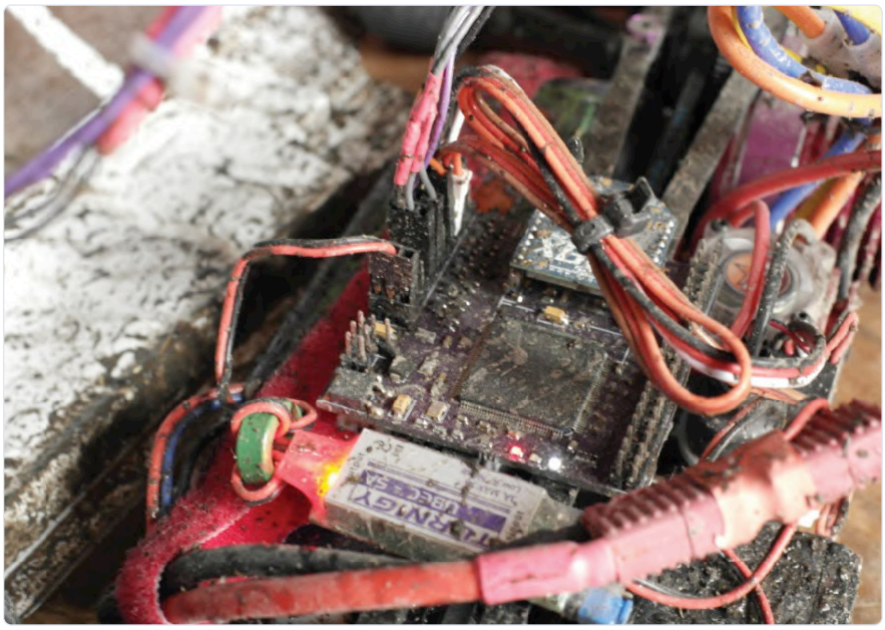


Figure 2. Disaster can happen!

One afternoon, while testing in an open park, the USB controller managed to briefly lose connection with the Arduino/ZigBee unit. This crashed the USB driver and issued full throttle to the radio, which kept on transmitting. The vehicle then propelled itself at full speed straight towards a crowded street and main road.

The ZigBee modules I'd used where long range, which meant that the vehicle would have to be well out of sight until the safety kicked in. As I quickly scrambled to rip the batteries off the Arduino, the vehicle flew past a crowd of people and smashed into a curb at what could have easily been 30-40 mph. Thankfully, no one was injured, and the impact resulted in the lithium battery pack being torn out the chassis, thus killing the power. Rather embarrassed and apologetic, I ran over to collect the smashed pile of carbon fibre and went home with my project in ruin (**Figure 2**).

I learnt that day that the simplest error can still seem perfectly normal to every other safety catch in a system. No matter how much they were engineered, one tiny little hack in between it all was enough to jeopardize the entire thing. Since then, I've taken failure mode effects analysis a lot more seriously and by principle apply it to every project I work on from the outset.

Modern tools and workflows, such as model-based design, allow for automated testing and coverage during development. This combined with an evolving release plan means that bugs are avoided early on or caught quickly in the release process. While things may occasionally still go wrong, a good test plan means that the same thing will almost never go wrong more than once. The images show the custom vehicle and control board. All the electronics where conformally coated to prevent dirt or water damage. A video on YouTube describes the project in more detail [1]."

The Necessity of Controlling High Current

Lars Krüger is a Potsdam, Germany-based teacher who has been reading Elektor magazine for eight years and designing electric bicycles since 1992. As you can imagine, he has learned a lot over the years about high current control, circuit testing, and e-bikes.

"In a former time and when I was younger, I tried to get things ready in a very short time. A quick idea in the morning had to be ready in the evening. I didn't care a lot about some malfunctions, when the rest was quite OK and running. But that led to a lot of broken and burnt semiconductors when it came to higher voltage and/or currents!

For instance, in 1992, I built my first electric bicycle. Having read only a quarter of a book, I prepared a PWM, running at some 100Hz, a set of 10 'BUZ' MOSFETs, a huge 100Ah car battery and the 2kW starter of a Citroen CX 2.5D. After the first 'throttling,' the motor was just 'on' (MOSFETs were burnt) and the bike, getting a speed of around 60kph+, was quite unstoppable. Of course, switching off the supply voltage of the PWM was useless because the burnt FETs didn't care about gate voltage. Lucky me that I built in a fuse and heavy braking lead to blow it. Next steps were that I built a large switch, made of

Figure 3. Krüger's e-bike projects taught him about high current control and more. Check out his e-bike from 1994.

thick copper and a spring. This was connected to the Bowden cable instead of the former potentiometer. The bike was just fun with this pure setup — 10m pedaling and then 'puuush.' Unable to control in the narrow streets of my hometown, but making me smile from one ear to the other when outwards.

What I learnt from that story was that controlling (limiting) current is a necessary thing, when scaling up the size of a motor, you can't skip or forget it. And that this control has to be fast enough to cope with the gradient of the current rise. Otherwise, you'll have oscillations in your control loop. Later on, I studied and heard 'control techniques,' all this 's+1 stuff' with minor effect. It had little to do with real life. A better experience were the upcoming simulation tools like PSpice or its follower SIMetrix. From that time on, I had much less problems with controlling high currents. But, of course, burning MOSFETs is not completely banned from my home lab. Sometimes, a probe slips off and produces a short while measuring. Good, if you're wearing goggles and the light circuit of the room is separated from the circuit you're testing at. By the way, the photo shows my second e-bike from 1994 (**Figure 3**). The first was too fast and seems that I had no time to take a picture of it."

MOSFETs, BJTs, and PLL Control

Robert Owen is a Stenløse, Denmark-based Senior System Engineer with years of experience in various areas of electronics. Here he shares details about what he learned while designing the sync separation and tracking control for CRT-based video projectors. After using a MOSFET transistor and building a board, he quickly ran into issues. Fortunately, he solved the problem without much difficulty and learned a bit about MOSFETs, BJTs, and PLL control.

SHARE YOUR ERROR ANALYSIS

Want to share details about your electronics-relate errors and learnings? Simply fill out our "Error Analysis" form. Professional electrical engineers, makers, and EE/ECE students are welcome to contribute.

www.elektormagazine.com/pages/error-analysis-submission

"Many years ago, in another lifetime, I was involved in designing the sync separation and tracking control for CRT-based video projectors. In order to accommodate users that insisted their $200 video recorder was perfectly displayed on our $10,000 projectors, we needed to be able to open the PLL control loop during the head switch interval from the video recorder. The control chip had a loop control input that would do the job. So, I popped in a MOSFET transistor to act as a variable resistor and built a new board. The circuit not only did not open the loop, but it appeared to try to track the sync signal even more closely resulting in image tearing at the top of the screen. (Our projectors displayed all lines of the image unlike a TV set, which cut off 10 or lines on each end.) What I discovered is that the control input was reacting to the current injection of the control input through that giant gate - source capacitor. Replacing the MOSFET with a JFET or in our case a simple BJT solved the problem without actually requiring and layout changes."

200303-01

WEB LINK

[1] **"Independent Wheel Drive 1/10 Scale RC Car - Project Aelith", SiliconWitch Couture, YouTube, 2014** : https://youtu.be/ET1HEyqEQJQ

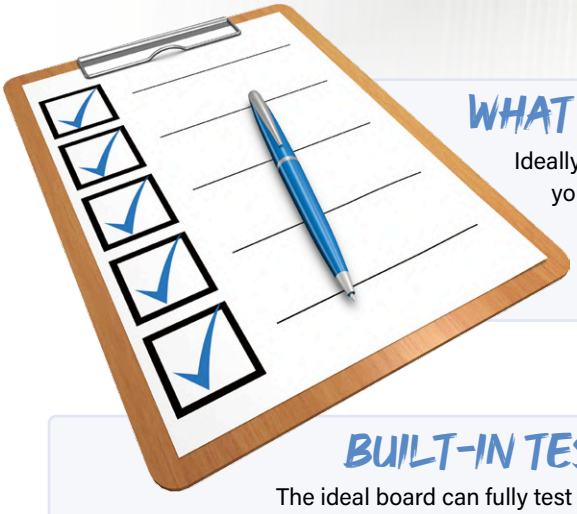
Developer's Zone

Tips & Tricks, Best Practices and Other Useful Information

By **Clemens Valens** (Elektor Labs)

FROM IDEA TO PRODUCT – PART 6

In the previous installment the electronics part of our new product advanced to the production stage and since then you have received a batch of boards. Now it is time to check that they work as intended, which implies testing. But how are you going to do that? Luckily, you have thought about this difficult subject way before launching production and so you come prepared.



WHAT MUST BE TESTED?

Ideally, every function or feature of the board must be tested. So how do you do this? How do you define a test? Do you test every board? Do you automate testing? Can it be automated? Would automated testing have been possible if only you had thought about it in an earlier stage of the product design? *Ouch!* Sorry, did that hurt?

BUILT-IN TESTS

The ideal board can fully test itself all by itself thanks to built-in tests (BITs). But even if it can do this, the test procedure must be activated, and the results interpreted, and somehow the failing boards must be separated from the good ones.

A built-in test can be implemented in software (don't forget to test the test), but not every board has software and software cannot test everything. A test routine that cycles through all the pixels or segments of a display is nice, but without an observer it is useless. To cite a famous dictum: '*If a tree falls in a forest and no one is around to hear it, does it make a sound?*'

BACK TO THE DRAWING BOARD

It is a recurring thing in this article series: every time you think you move a step forward, you end up going back to the product specification stage. It is the same here. Board testing should be part of the design specifications. If you think about testing from the start, then you can adapt the design to make it as easy and complete as possible. Always keep in mind that, in the end, testing is done by people or machines that have no knowledge whatsoever of the device under test (DUT). Therefore, the simpler the test procedure, the better.

TEST CONNECTORS AND TEST POINTS

Adding a test connector to the board is a possible solution. JTAG for instance is a popular method for testing the connections of logic devices like microcontrollers and FPGAs to the PCB and also between the devices.

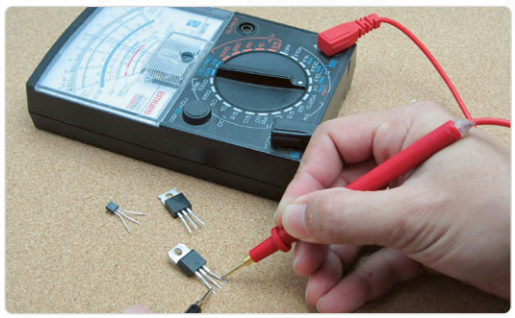
A test connector can expose inputs and outputs, analogue, digital, or both that can be excited and read by a tester, be it a human or a machine. The tester can inject key presses and monitor a test output signal. However, injecting a key press does not tell you if the key itself is present *and* working. Also, a test connector eats up space ('board real estate') and costs money, not counting any extra components required to make it work.

To work around this, designers often use test points instead. Oscilloscopes and similar instruments can be programmed to check if (complex) signals on the test points stay within boundaries. The advantage clearly is the small size of test points and the freedom of placement, but a special test jig is needed to connect to them. Also, the test instruments need programming and test jigs must be developed, and tested and, obviously, all this costs time and money.

The quality of your test procedures can make all the difference between relaxing on a tropical island and being blown out of business.

DESIGN AN EXTENSIVE TEST PROCEDURE

No matter how testing is implemented, a clear test procedure must be defined that results in a reliable Pass/Fail decision. The procedure should cover every aspect of the board and must be as easy to implement, understand and execute as possible, and even more so when production quantities increase. Designing such a procedure is not something to take lightly. As said above, in the end, the tester should require no or little training, not only to keep the costs down but also to minimize the risk of mistakes and errors during testing.



The method of testing not only depends on the number of products to test.

ENDURANCE TESTING

Functional testing alone is nice, but may not be good enough. Also think about endurance testing. You don't have to do this for every board, but it is good practice to take a few samples from each batch, visually inspect them thoroughly, and put them in a climate chamber for temperature cycling and humidity testing. Do the samples survive? Depending on the application of the end product other endurance tests like drop and shock tests might be required too.

COMPLIANCE TESTING

There is also compliance testing to take care of and prepare for. Now this is easier in the sense that you don't have to come up with test procedures as these have been defined by the authorities (even though you may have to instruct the compliance testing agency how to correctly manipulate your product), but the hard part can be making your product compliant. Again, this costs time and money. Not only do you have to pay for the compliance testing itself, you must also pay someone to figure out which norms concern your product and in which country, and what the implications are for your product. It may be interesting to invest in some pre-compliance test tools so that you can detect and correct potential problems early on. And, once again, preparing for compliance testing too should have been done before you started spending any real money on your product.

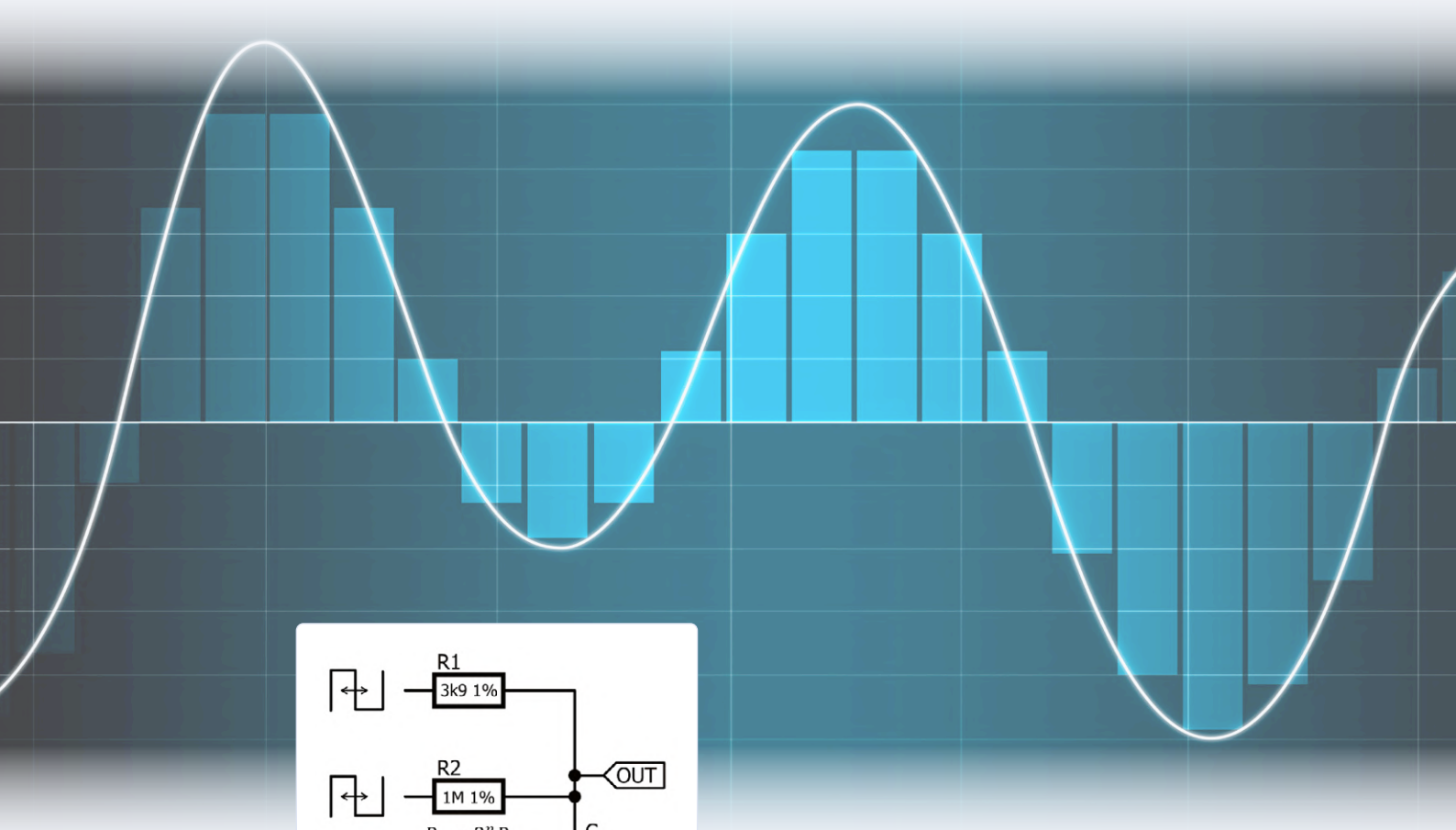
LISTEN TO FEEDBACK

It is also good practice to listen to feedback from the people that execute the tests you specified as it may help to improve the board. This may result in for instance a better quality and end-user experience, fewer returns, cheaper production and/or faster testing.

TESTING COSTS MONEY

Since imagining, implementing and executing good test procedures is an integral part of product design and manufacturing, it requires time and effort and therefore money. Unless you live in a perfect world, testing will also generate a stream of failing boards. They can simply be discarded, or you might try to repair them. Whatever you decide to do, here too is a price to pay. The only way to lower these costs is to improve the quality of your board and the way it is produced. This is where quality engineers come in. If you try to reduce the number of failed devices by simplifying the test procedure, you can count on being bitten later by aftersales. And that is the whole point of product testing: avoiding unhappy customers complaining about their products and returning them, and giving you a bad reputation. In the end, dealing with aftersales is way more expensive than trying to do it right from the start. It can even cost you your business.

Dual PWM
Improves 8-bit Audio Quality



A digital-to-analogue converter or DAC transforms digital values or numbers into an analogue voltage proportional to the digital input value. The output resolution is determined by the number of bits the DAC can handle, its word width so to speak. To increase the resolution one can simply take a DAC with a larger word width. Another solution is to put two low-resolution DACs in parallel. If we use one DAC to produce a coarse value, a second DAC can then fine-tune it ('fill in the gaps') if we scale its output properly. This way, theoretically, one can obtain 16-bit resolution with two 8-bit DACs. In practice the resolution is less, however, because the scaling will be imperfect unless very precise electronics is used. This would be more complex and expensive than simply using a 16-bit DAC, and so this technique is not used very often.

Most microcontrollers (MCUs) do not feature a DAC, and common practice is to use pulse-width modulation (PWM) to create analogue signals. Low-pass filtering a PWM signal results in a voltage proportional to the pulse width. The resolution is now determined by the resolution of the PWM signal. Arduino, for instance, has the `analogWrite()` function to generate PWM signals with a resolution of 8 bits. The Arduino Uno can output up to six of these signals at the same time (on the pins labelled with a tilde '~').

We can apply the same trick as described above to increase the resolution: let one PWM output produce the coarse value and use another for fine-tuning it. Scaling can be done with the resistors that are part of the low-pass filter. The resulting circuit is simple. Of course, the precision of the resistors is important, making real 16-bit resolution difficult to obtain, but with 1% types the usable resolution will be almost 14 bits.

But, I hear you say, the MCU on the Arduino Uno can also do 16-bit PWM, so why not use that instead? The reason is speed. For the same clock frequency, 16-bit PWM will have half the output rate of 8-bit PWM. A high output frequency or sample rate is interesting for e.g. audio applications.

So, why not put three PWM signals in parallel, or even more? You can try, but at some point, you will run into software speed limits and hardware complexity issues. In short, the required effort is probably not worth it. Dual 8-bit PWM is a good compromise. A good explanation and analysis of dual PWM can be found at:

www.openmusiclabs.com/learning/digital/pwm-dac/dual-pwm-circuits/

Siglent SDL1020X
Programmable DC Electronic Load

By **Harry Baggen** (The Netherlands)

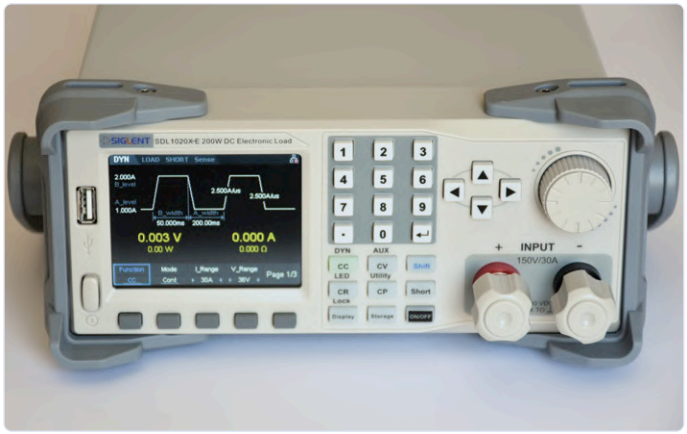
For testing power supplies, batteries and accumulators, an electronic load is a very useful tool. Such a load not only imitates a fixed resistance; it can also act as a constant current or voltage load and can even switch between different load values. The Siglent SDL1020X-E offers a whole range of such capabilites and is suitable for voltages of up to 150 V and currents of up to 30 A, with a maximum dissipation of 200 W.

When designing or testing a power supply circuit, it is important that it remains stable under various types of loads, reacts quickly to load changes, and outputs low levels of interference. In recent years there appears to be an increasing need for a device that can load a power source in different ways and, especially from Chinese manufacturers of measuring equipment, we have see the introduction of quite a few electronic loads. Siglent has also had a series of electronic loads in its programme, the SDL1000X series, for less than a year now. In this review I have tested the SDL1020X-E. This is the smallest version that can handle a maximum power of 200 W. There is also a 300 W version and for both versions there are models without the -E suffix that offer a higher reading accuracy.

Solid case

When discussing Siglent equipment, I've mentioned previously that they are usually equipped with a robust housing and this is also the case here. What is particularly striking is the length of the housing, namely 39 cm. If you want to put the device on a shelf, you'll have to make sure you have enough space.
The front panel contains all control buttons, two heavy-duty

The cabinet is a lot deeper than most other Siglent devices.



The front panel with the display showing the setting for dynamic waveforms.

The rear panel contains a range of attachments and a number of cooling slots.

The large screw clamps are very sturdy, but unfortunately do not permit the use of banana plugs.

connection terminals, and a large 3.5" LCD display showing all the required operational information. Below the screen there are 5 buttons whose current function is displayed on the screen.

At the rear we find the power connection; a USB, LAN, and RS232 connector; and two BNC sockets that provide a measurement signal for the voltage and current of the load (for connection to an oscilloscope, for example). At the bottom is a connector block with a whole series of terminals of which I will only mention the most important ones here, namely the sense inputs and trigger input. To the left are air slots for cooling the large cooling tunnel that runs internally along almost the entire length of the device. At the front is a temperature-controlled fan. At low loads the SDL1020X-E is therefore virtually silent.

Options

The SDL1020X-E offers so many possibilities that a few review pages are not enough to cover everything. I will limit myself to the most important things. Should you wish to purchase this device, it certainly makes sense to examine the datasheet and manual in detail.

Let's start with the basic functionality. You can choose between constant current (CC), constant voltage (CV), constant power (CP) or constant resistance (CR) modes. The display always shows the measured voltage, current, power, and resistance. There are two current ranges (5/30 A) and two voltage ranges (36/150 V) that are mainly related to measurement accuracy. A separate battery test function allows, amongst other things, discharging at a fixed current, resistance, or power down to a set threshold voltage, whereupon the display shows the discharge capacity and power.

Most interesting are the dynamic test capabilities that are not so easy to implement without such specialised equipment. When the dynamic test function (DYN) is activated, a block-shaped waveform appears on the display. This indicates that you can let the instrument switch between two values using parameters you define yourself. These include the low load value for CC/CV/CR/CP, their high value, low time, high time, and steepness of both the rising and the falling edges. This can be configured at frequencies of up to 25 kHz, which will certainly be sufficient for most test

The test setup.

situations. The dynamic test can also be triggered from an external or manual signal.
Using the List and Program functions the user can compose more complex load waveforms in a number of steps. This offers an extremely powerful testing capability. With the List function (max. 100 steps per waveform) the load type (e.g. constant current) is identical at all steps; with Program (max. 50 steps per waveform) you can also change settings at each step, e.g. from constant current to constant resistance. With both functions you enter all the required parameters for each step in a tabular interface.
By default all measured values are shown on the display and, in dynamic mode, the square wave with all set values appears. If you switch to DISPLAY mode, the change of a selected parameter is shown graphically over time.

Practical use

After connecting a lab power supply, a scope, and an accurate multimeter, I was able to start testing this electronic load. The accuracy of the displayed values was very good and usually within 1 mA/mV. The large connection terminals provide enough space to firmly connect cables directly or using cable forks.
Having tried out some simple tests I turned to the manual because of the huge number of settings. So far I have only mentioned a small collection of what is possible. Through the manual I also found out that the two monitor outputs at the rear need to be activated in a submenu before they provide measurement signals. Unfortunately, this setting is reset when the device is switched off and must be reactivated the next time.
I also discovered that the maximum steepness of the edges in the dynamic test mode depends on the selected current range. Thus, in the 30-A range it is 2.5 A/µs and in the 5-A range only 0.5 A/µs. The large screw clamps are very sturdy but, unfortunately, do not permit the use of banana plugs. This is one thing I miss here as they are quite convenient when working with currents up to a few amperes.
Apart from these trifles (most of which can probably be adjusted with a firmware update) the SDL1020X-E is a fine device to work with and maintains the quality we have come to expect from Siglent. It is also possible to control the electronic load via your PC, but this is currently only possible via SCPI commands or a LabView driver. Siglent still seems to be working on a Windows program, but it is not known when that will be available.

Conclusion

If you frequently need an electronic load for testing power supplies or batteries, a device such as the SDL1020X-E is an ideal addition to your lab instruments. It offers almost all the testing options you can think of. The clear display shows all the necessary information and makes connecting a separate oscilloscope or multimeter unnecessary in many cases. Overall, this is a well thought-out instrument at an affordable price!

200323-04

SHOPPING LIST

› **Siglent SDL1020X-E Electronic Load**
www.elektor.com/siglent-sdl1020x-e-programmable-dc-electronic-load-200-w

High-Voltage Power Supply with Curve Tracer

Generate voltages up to 400 V and trace characteristics curves for valves and transistors

By **Rainer Schuster** (Germany)

This universal high voltage power supply is not only capable of providing power to valve circuits; it can also be used to trace the characteristics curves of valves and semiconductors. The unit is controlled by a Raspberry Pi equipped with a seven-inch touchscreen.

Pretty much every hobbyist will have a power supply on their bench. On the list of indispensable lab tools, it ranks alongside the soldering iron and the multimeter. The most common type is a relatively compact device with an output voltage range of up to 60 V, and at most a single-digit current output. This is more than enough for most garden-variety circuits that one may want to power in the lab. But sometimes we encounter a more exotic species that requires higher voltages, and this is especially the case when working with valve circuits.

When constructing a power supply dedicated to higher-voltage applications we can use

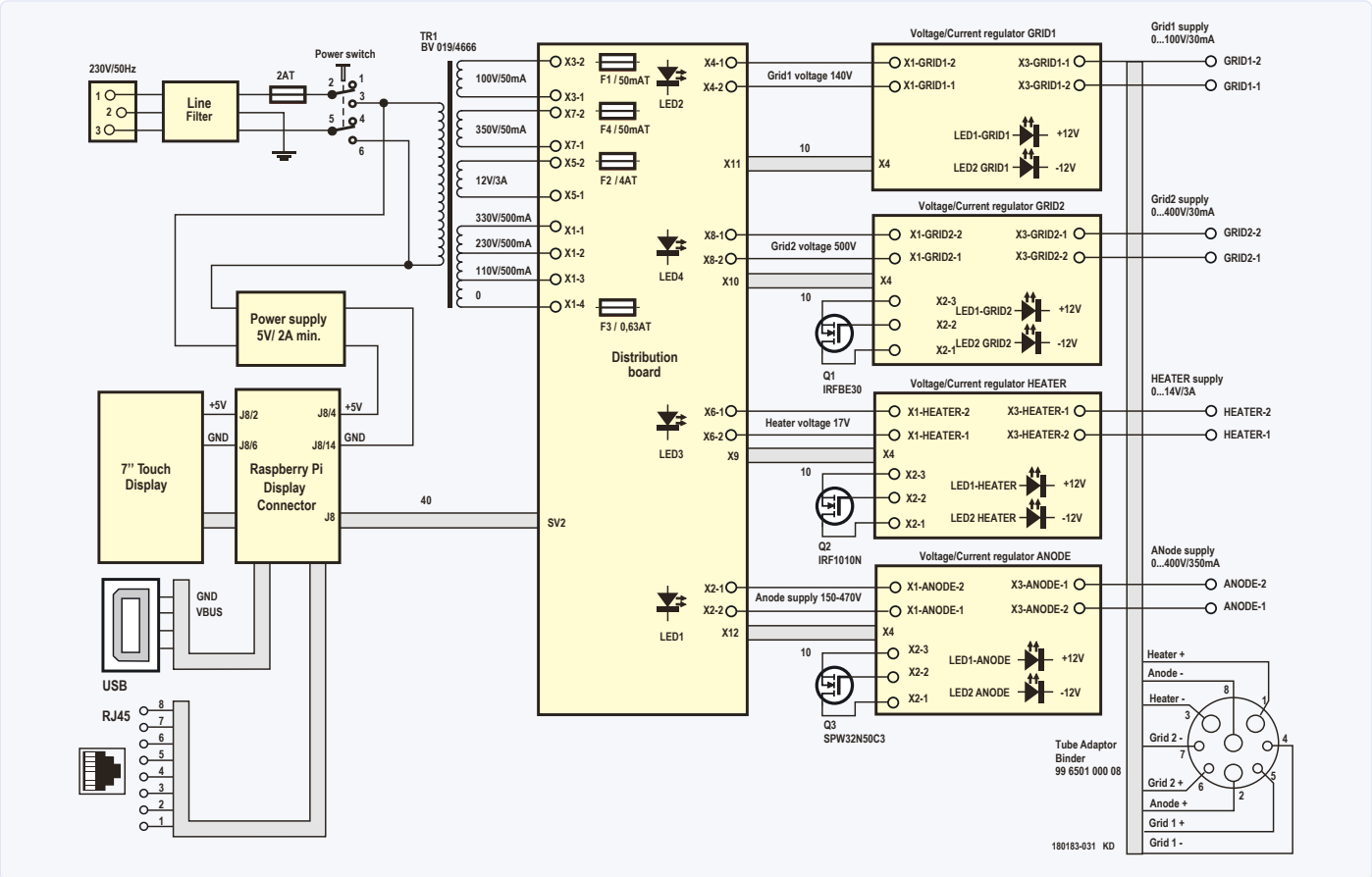


Figure 1. The block diagram of the high-voltage power supply gives an overview of how the various parts are connected together.

modern technology to add a couple of extra useful functions. And by 'modern technology', we mean a Raspberry Pi. Our application is not particularly demanding, so we do not require the most recent Pi (version 4); a version 2 or version 3 model are perfectly satisfactory. The user interface and display are given a modern touch with the help of a seven-inch touchscreen. Useful additional functions, such as the ability to trace characteristics curves for valves and semiconductors, are also implemented in software.

Power supply hardware

The power supply is of course capable of generating more than just a 'high voltage' output. (We note in passing that, strictly speaking, 'high voltage' normally refers to voltages over 1.5 kV and our supply's limit is much lower than this.) To power a valve we need not only the high anode voltage but also a further supply for the heater. If we additionally want to trace its curve, further voltages are needed to apply to the control and screen grids. In our design all these voltages and their corresponding current limits are adjustable.

The hardware to regulate the voltages and currents of the individual outputs consist of four independent supplies that are galvanically isolated from one another. They are capable of generating the following outputs:

- Anode: 0 to 400 V, 0 to 300 mA
- Control grid: 0 to 100 V, 0 to 30 mA
- Screen grid: 0 to 400 V, 0 to 30 mA
- Heater: 0 to 14 V, 0 to 3 A

Figure 1 shows the block diagram of the power supply and the wiring between its individual modules. The core functionality of the supply is carried performed by the distribution board and the independent regulator modules.

Distribution board

The distribution board has the rectifiers, smoothing capacitors and fuses for the various regulator boards. SV2 is a 40-way header that connects to the I/O ports of the Raspberry Pi and the SPI signals are distributed to the regulator boards via X9 to X12.

In the interest of reducing power dissipation in the output pass transistors, the transformer winding used to generate the anode voltage is switched between 110 V, 220 V and 330 V using relays K1 and K2. These relays are controlled in software using GPIO pins 20 and 21 of the Raspberry Pi. **Figure 2** shows the circuit diagram of the distribution board.

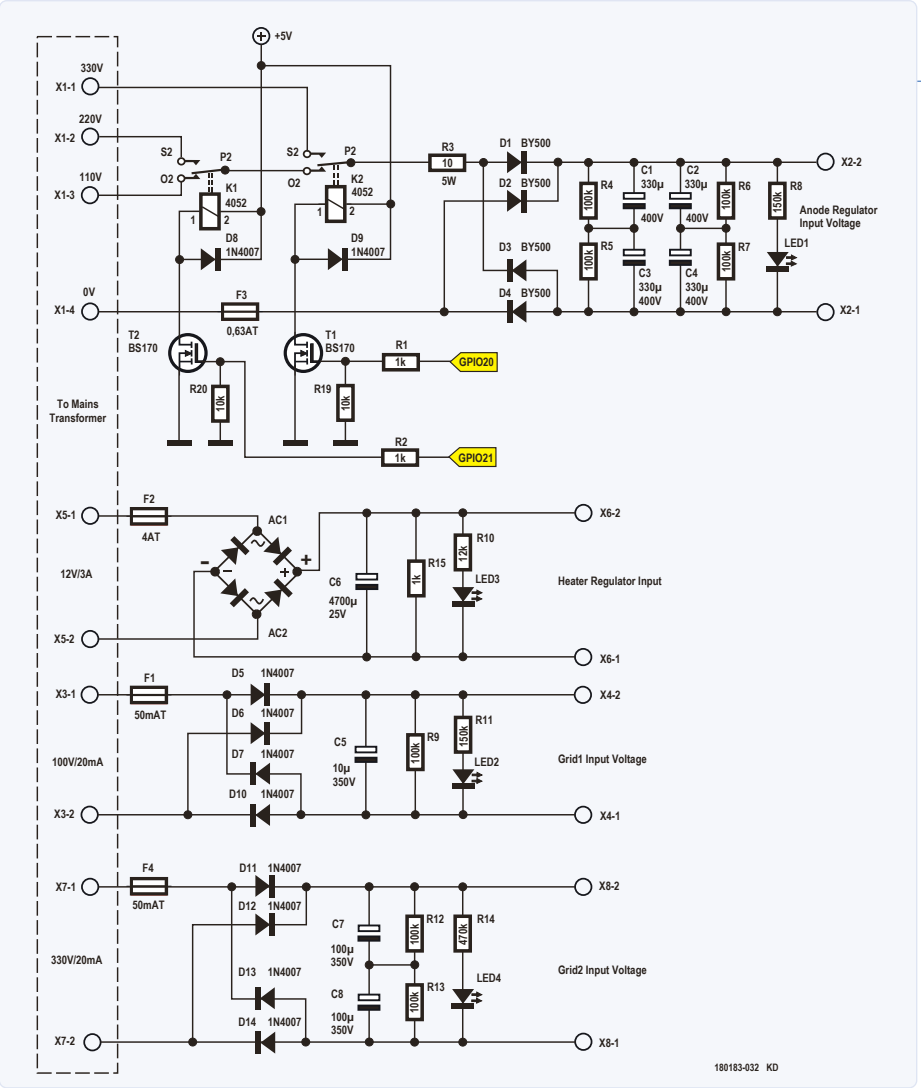


Figure 2. Circuit diagram of the distribution board.

Regulator boards

Each regulator board contains A/D and D/A converters to set and measure the output voltage and current. The Raspberry Pi communicates with these converters using its SPI interface. The SPI signals are galvanically isolated from the Raspberry Pi.

Circuit operation

Figure 3 illustrates the principle of operation of the regulator circuit. MOSFET T2 is configured as a series regulator. Since we are dealing with high voltages and T2 is controlled by an ordinary op-amp powered from a ±12 V supply, the positive terminal of the output voltage is connected to the ground reference of the op-amp.

An MCP4812 two-channel DAC with 10-bit resolution is used to control the voltage and current set points. This device contains an internal 2.048 V voltage reference which can be doubled to 4.096 V using an SPI command. The output voltage is divided down by R17 and R18 with the voltage across R18 being digitised using an LTC1298 12-bit A/D converter and sent back to the Raspberry Pi. For current regulation, the voltage across R14 is multiplied by five using an op-amp. This voltage, which represents the output current, is sent to the Raspberry Pi using the second channel of the A/D converter.

6N137 high-speed optocouplers are used to isolate the MOSI, MISO, SCLK and LDAC SPI signals and the chip select signals for the A/D and D/A converters. The optocouplers are driven using a 74HC04.

Of course, each regulator board also needs its own galvanically-isolated internal power supply. A TMA0512 DC/DC converter generates the ±12 V supply required by the op-amps from the 5 V output of the Raspberry Pi board.

As well as being fed to the inputs of the A/D converters, the voltages representing the instantaneous voltage and current output of

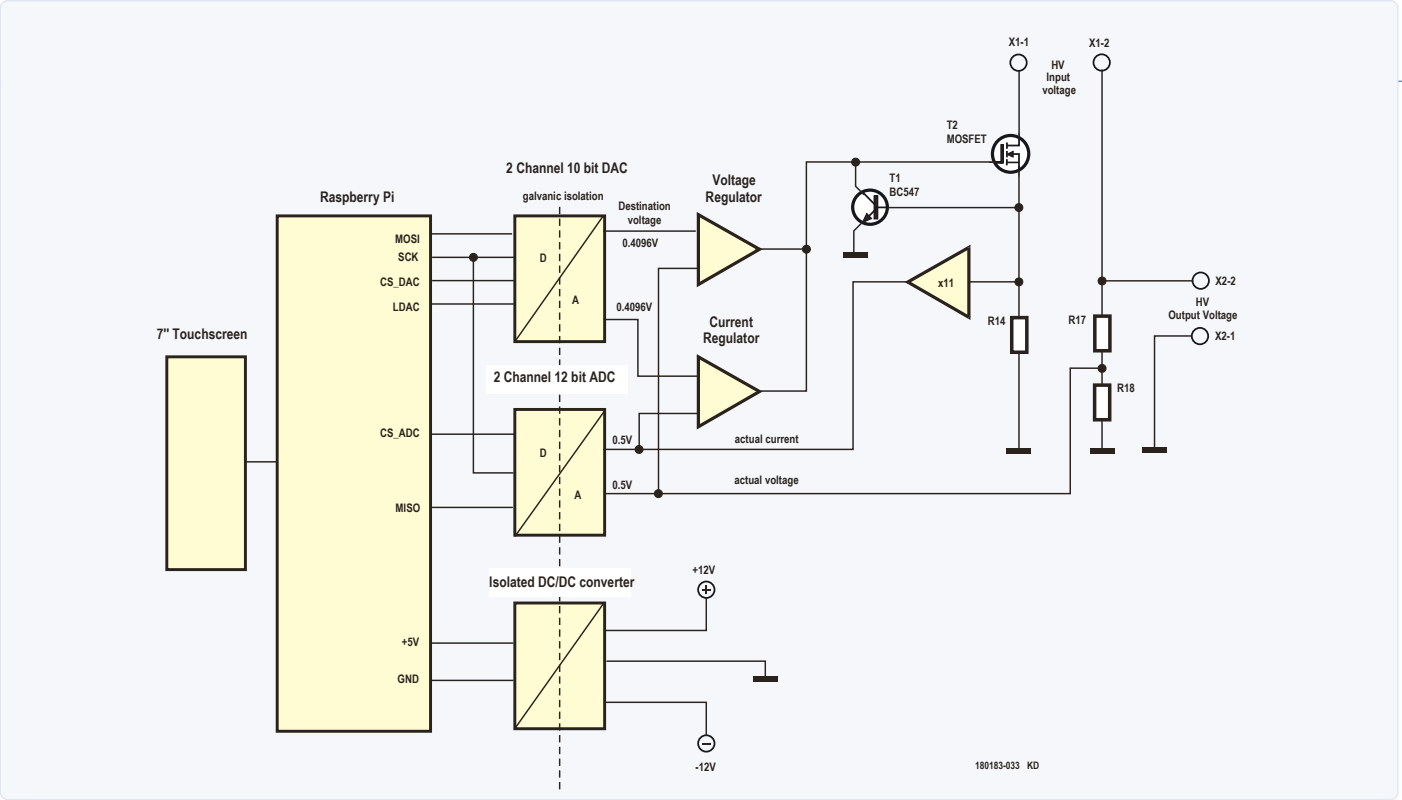


Figure 3. Block diagram for the high-voltage power supply circuit.

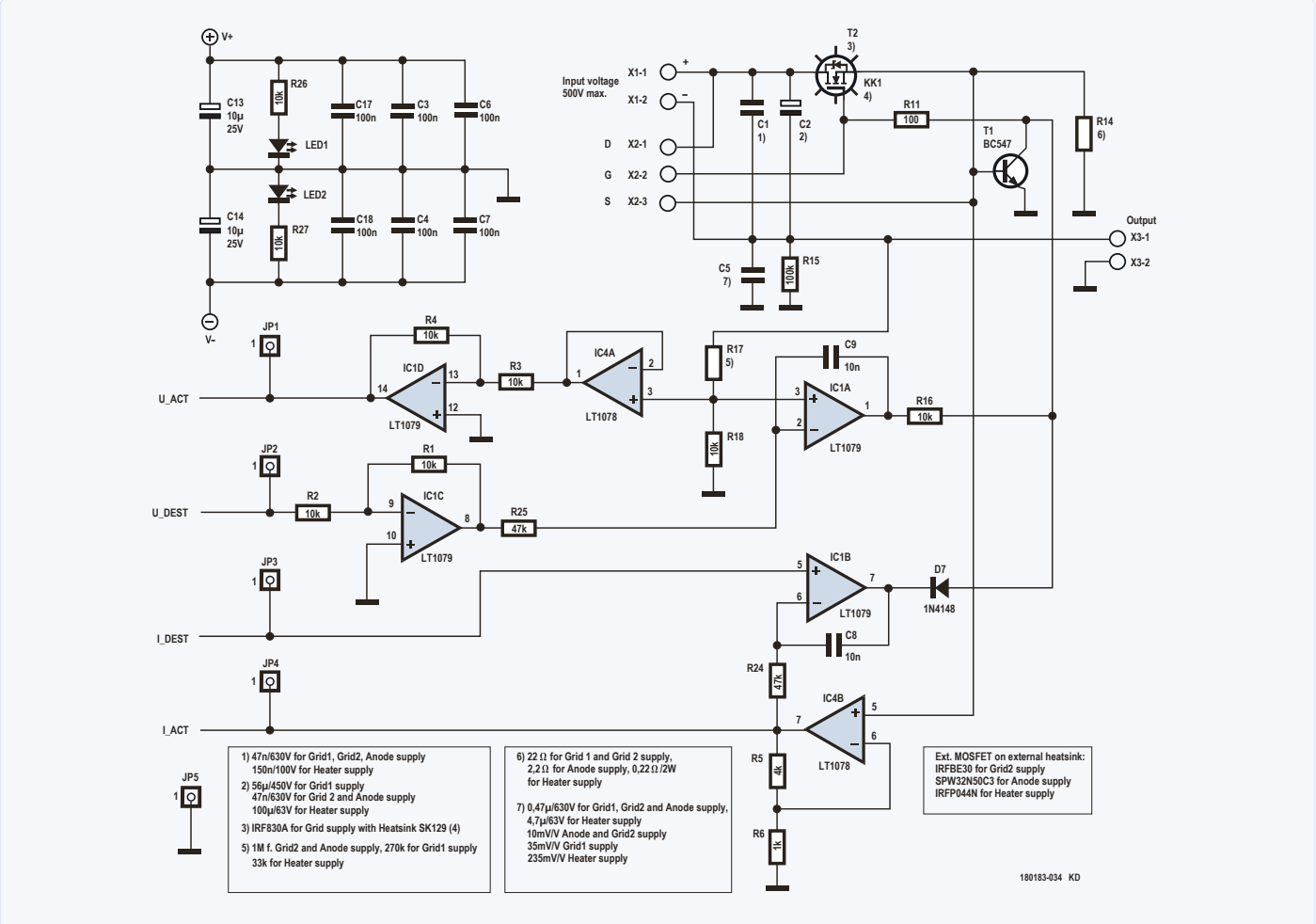


Figure 4a. Circuit diagram of the regulator board.

the supply are also used to provide feedback to the two op-amps used for voltage and current regulation. These op-amps compare these instantaneous values with the set point voltages produced by the D/A converters. The correction signal that is produced by the voltage regulation op-amp is then used to drive the gate of T2. Current regulation comes in to play when the measured current is higher than the set point. The supply therefore provides constant voltage regulation with an adjustable current limit.

The current regulation behaviour of the supply means that, if there is a short circuit across the output, the current delivered will briefly overshoot the set point. In the case of the anode voltage supply board this could involve T2 being taken outside its SOA (safe operating area). To prevent this, T1 is used to quickly clamp the current to a maximum of 0.7 V / R14 = 0.31 A. The power dissipation will then briefly hit 465 V * 0.31 A = 144 W until the software switches the input voltage over to the 110 V winding of the transformer. According to its datasheet, T2 can cope with nearly double this power dissipation.

Figure 4 shows the circuit diagram of each regulator board. Readers interested in the regulation behaviour can review the LTSpice simulation that was performed. The file, *HVRegulator.asc*, is included in the archive and can be freely downloaded from the Elektor web page accompanying this article [1]. All the regulator boards have the same layout but are populated with different component values to suit the different voltage and current ratings (see **Table 1**).

Raspberry Pi

As mentioned, the user interface for the power supply is implemented with a Raspberry Pi and a seven-inch touchscreen. This part of

Table 1.

Board	C1	C2	C5	T2	R14*	R17**
Anode	47 nF 630 V	47 nF 630 V	0,47 uF 630 V	SPW32N50C3 (HS ≤0,7 k/W)	2,2 Ω 1 W	1 MΩ
Control grid	47 nF 630 V	56 uF 450 V	47 nF 630 V	IRF830A (KK SK129)	22 Ω	270 k
Screen grid	47 nF 630 V	47 nF 630 V	47 nF 630 V	IRFBE30 (KK ≤0,7 k/W)	22 Ω	1 M
Heater	150 nF 100 V	100 nF 63 V	4,7 uF 63 V	IRFP044 (KK ≤0,7 k/W)	0,22 Ω 2 W	33 k

* R14 = 4.096 V / (5 * I_{max})
** R17 = U_{max} * 2.441 kΩ / V – 10 kΩ
HS = Heat Sink

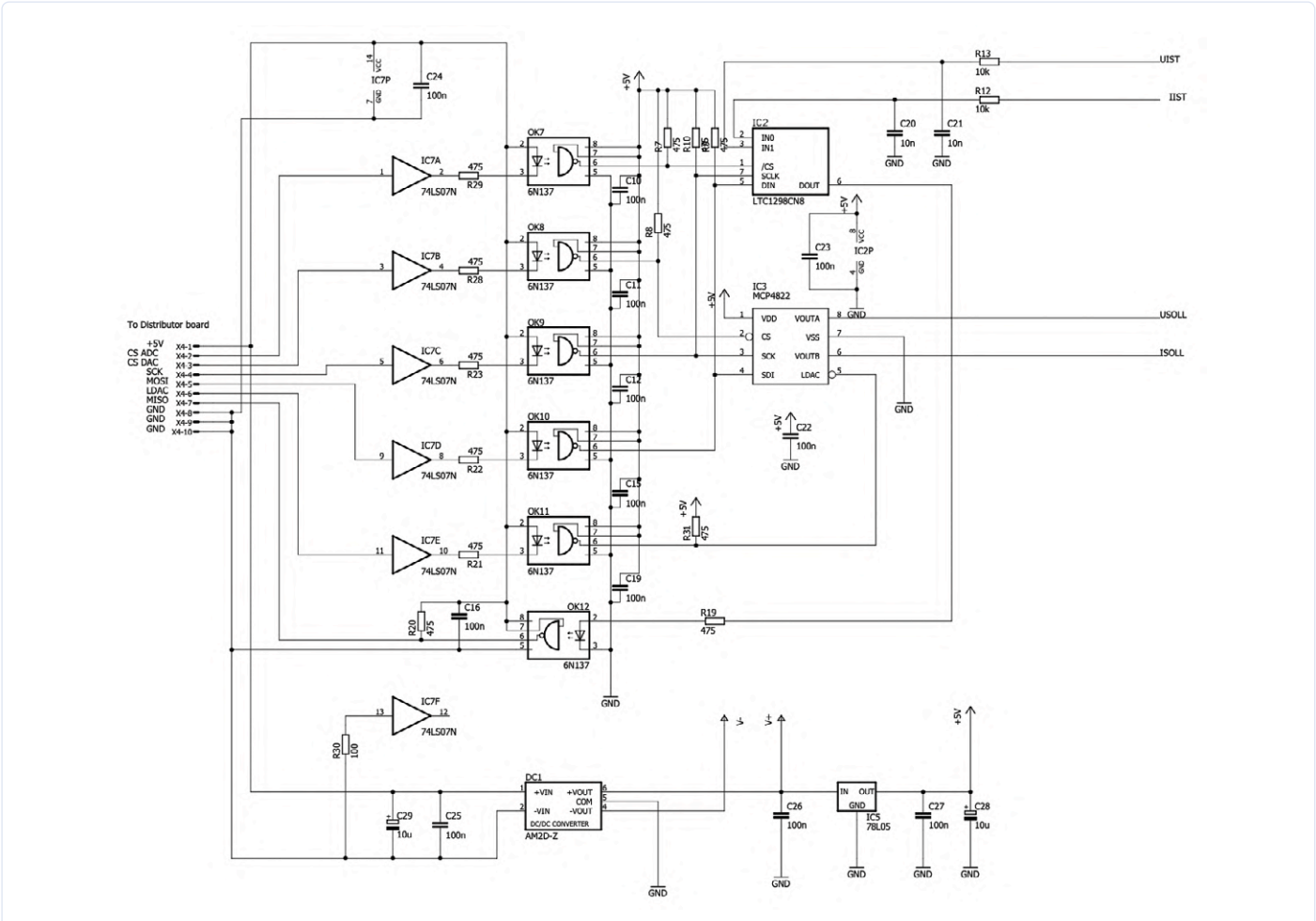


Figure 4b. Connection of the regulator board to the distribution board.

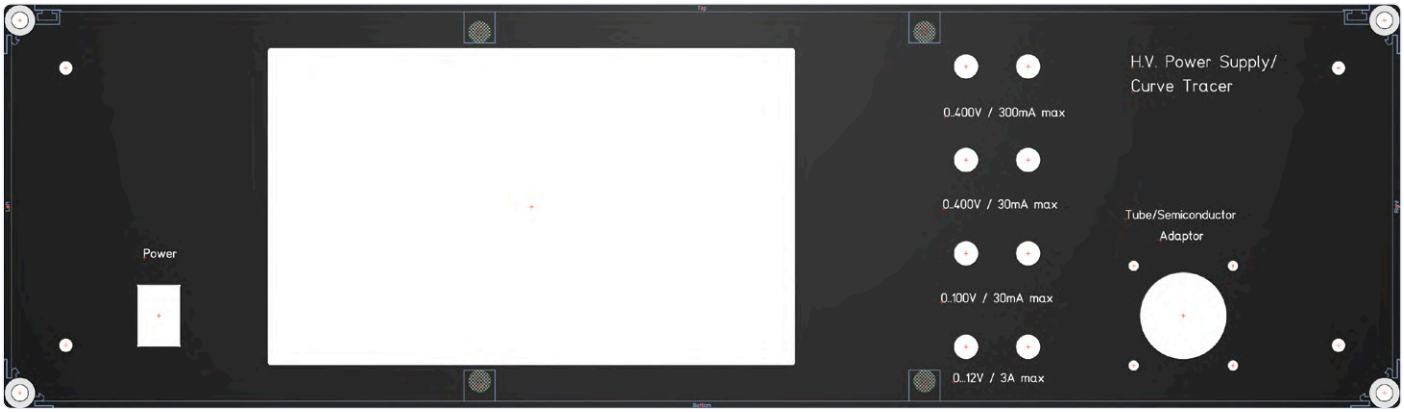


Figure 5. The front panel was designed using the program FrontDesign.

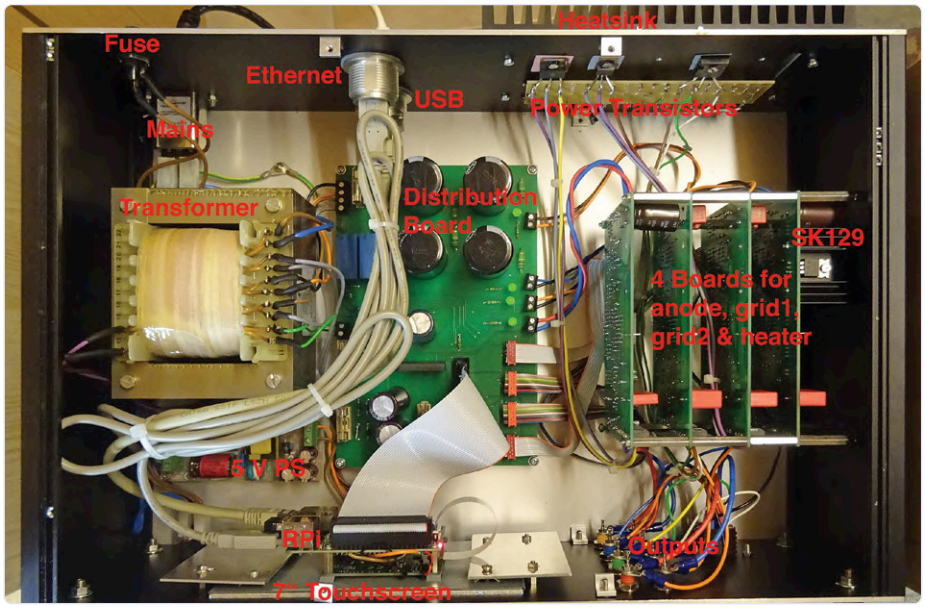


Figure 6. Internal construction of the prototype.

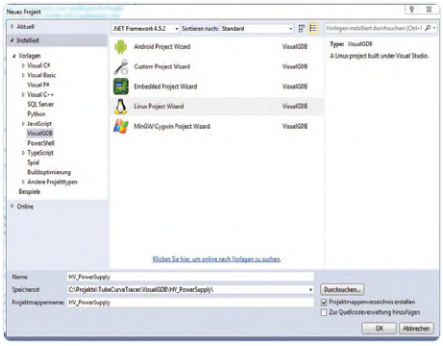


Figure 7. Configuration of VisualGDB for a Linux project.

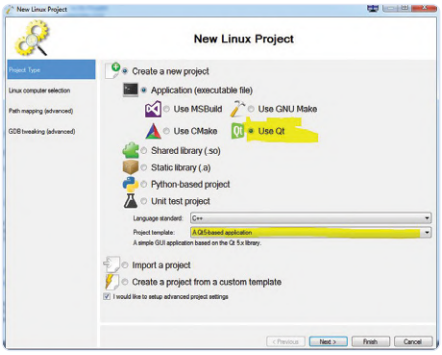


Figure 8. The program uses the Qt5 toolkit.

the unit can be powered by any open-frame supply capable of delivering 5 V at 2 A (or 3 A for a Raspberry Pi 4). As the voltage drop via the micro-USB connector can be high, it is best to feed the power directly using header J8.

Construction

The unit can be built into an extruded aluminium enclosure designed with the help of FrontDesign, which is available free of charge from Schäffer [2] for Windows, MacOS and Linux. **Figure 5** shows the design of the front panel and the corresponding design files are available to download from the Elektor website [1]. **Figure 6** shows the construction of the prototype from above. The four similar-looking regulator boards can clearly be seen on the right-hand side.

Software

The software running on the Raspberry Pi was written using C++ in Microsoft's Visual Studio [3]. The VisualGDB cross-compiler [4] is required to build code that can run on the Raspberry Pi. A free test version of this tool is available that functions for 30 days. Qt Designer [5] was used to create the graphical interface.

The download [1] includes the project file *HV_PowerSupply.sln* that contains all the required settings for VisualGDB (see **Figure 7**). These include that information the project is to run under Linux, the name of the project, and the folder where the project data will be stored.

In the next step we configure the project to use Qt5 (see **Figure 8**). We then have to decide whether to build the project locally on the PC or remotely on the target system (see **Figure 9**). In this case I have chosen to build the project on the local PC as other-

wise it is necessary to keep the Raspberry Pi turned on and connected to the PC throughout the compilation and linking process. In the window shown in **Figure 10** we specify that the WiringPi library [6] is to be used.
Once the software has been built successfully you should find the file *HV_PowerSupply*, an executable ready to run on the Raspberry Pi, in the project's *Debug* folder. This file can now be copied to the directory */home/pi/HVPowerSupply* on the Pi using, for example, WinSCP [7]. The project icon *Tube.png* should also be copied into this directory.
In order to allow the application to be launched from the desktop, the file *HV_PowerSupply.desktop* should be copied into */home/pi/Desktop*. If you would like the program to run automatically on power up, this file should also be copied to */home/pi/.config/autostart*.
To ensure that the up and down symbols are correctly displayed on their buttons, the files collapse-*arrow.png* and *expand-arrow.png* should be copied into the */home/pi* directory. If you would like the valve icon to appear in the taskbar, copy the file *Tube.png* into the */home/pi* directory as well.

If you are happy to use the software unmodified you can of course simply copy the pre-compiled files from the download archive directly to the Raspberry Pi.

Source code

If you wish to modify the software then you will need to install the package mentioned above and copy the source files into the */VisualGDB/HV_PowerSupply/HV_PowerSupply* folder. The project is comprised of the following source code files.

- *HV_PowerSupply.cpp*: main program, which calls *MainWindow.cpp*.
- *MainWindow.cpp*, *MainWindow.h*, *ui_MainWindow.h*: source code for the power supply functions of the device.
- *CurveTracerWindow.cpp*, *CurveTracerWindow.h*, *ui_CurveTracerWindow.h*: source code for the valve curve tracer functions.
- *TransistorCurveTracerWindow.cpp*, *TransistorCurveTracerWindow.h*, *ui_TransistorCurveTracerWindow.h*: source code for the semiconductor curve tracer functions.
- *Hardware.cpp*, *hardware.h*: common code for interfacing with the D/A and A/D converters using the WiringPi library.

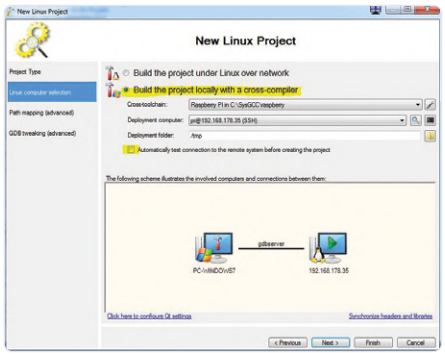


Figure 9. The software can be built locally on the PC or remotely on the target Raspberry Pi system.

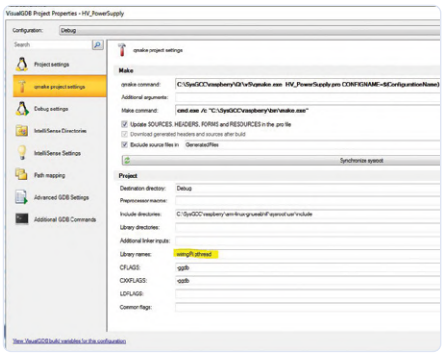


Figure 10. Defining use of the WiringPi library.

- *Qcustomplot.cpp*, *qcustomplot.h*: graphics library for plotting characteristics curves [8]. To ensure that this file is compiled correctly, the line `QT += core gui xml printsupport` must be added to the file *debug.pro*.

The file names following the structure *ui_XXXX.h* correspond to the graphical interfaces for the various screens and were created using Qt Designer. **Figure 11** shows the screen for controlling the power supply functions. The source code is produced with the help of the *Form -> View Code* menu item and can then simply be saved into the project source directory.

Program execution

When the program is started the hardware will be initialised and the power supply control

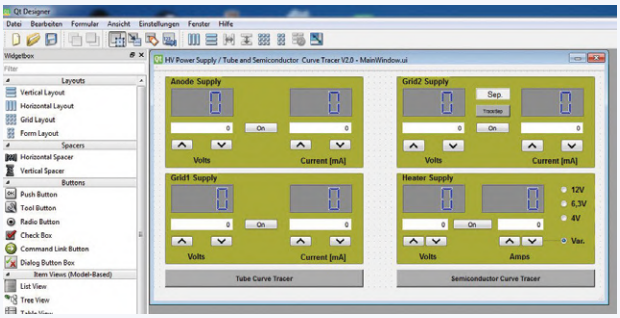


Figure 11. The high-voltage power supply control screen in QtDesigner.

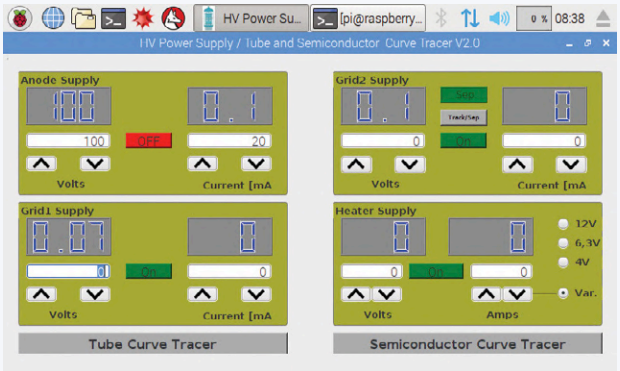


Figure 12. The user interface for the high-voltage power supply functions.

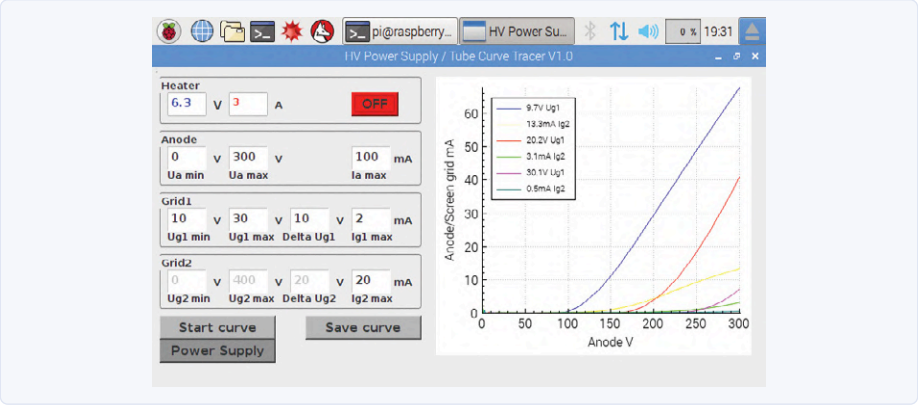


Figure 13. The curve tracing function user interface for a valve.

screen will be displayed. All voltage and current settings are initially set to zero. You can now adjust the voltages and currents for each output individually using either the up/down buttons on the touchscreen, an external keyboard and mouse connected over USB, or a virtual keyboard such as the 'Matchbox Keyboard' [9] (**Figure 12**).

The screen grid voltage can be made to follow the anode voltage by enabling tracking mode using the 'Track/Sep' button. The heater supply can be adjusted between 0 V and 14 V using 'Var' mode, or it can be set to one of the preset values of 4 V, 6.3 V or 12 V. The 'On/Off' button allows each output to be individually turned on or off.

Valve curve tracer

The button labelled 'Tube Curve Tracer' activates the corresponding user interface screen. All voltages and currents are reset to zero.

In this mode it is possible to record the characteristics curve of a valve and, if required, export the data as a table in CSV format so that it can be imported into a program such as Excel. To generate the curve it is necessary to specify minimum and maximum values for the anode and control grid voltage, as well as maximum current settings for the anode, control grid and screen grid outputs. The screen grid voltage tracks the anode voltage. A maximum of five characteristic curves can be displayed.

Figure 13 shows the curves recorded from an EL34 valve with the anode current as a function of anode voltage and the control grid voltage as a parameter. The corresponding screen grid current is also displayed.

Curve data is stored using the 'Save Curve' button. A dialogue box will appear prompting you to save the curve data in a file with a '.csv' extension. The resulting file can be copied to your PC using a program such as WinSCP, allowing the data to be analysed using Excel. The download archive [1] includes the example file *EL34.csv*.

To test the EL34 output amplifier valve I built an adaptor that allows the device to be simply connected to the curve tracer with a cable (see **Figure 14**). The straightforward circuit of the adaptor is shown in **Figure 15**.

By using a different valve holder it is possible to build a modified adaptor to suit different types of valve. Alternatively, it may be possible to connect to the pins of the valve directly using crocodile clips.

Semiconductor curve tracer

The button labelled 'Semiconductor Curve Tracer' brings up the user interface screen for this mode of operation. **Figure 16** shows an example of a reverse-biased bipolar transistor with its base unconnected.

Again, an adaptor comes in handy when testing semiconductors. **Figure 17** shows the circuit of the semiconductor adaptor and **Figure 18** the finished unit. **Figure 19** gives a glimpse inside.

The curve tracer is connected to the header SV1. The heater supply is used to operate relays K1 and K2 that switch between NPN and PNP (or NMOS and PMOS) modes. The control grid voltage is used to operate the relay K3 that switches the base current (or the gate voltage). The anode voltage is connected to the collector/emitter (or drain/source). To test a Zener diode, the negative side of the anode voltage should be connected to the anode and the positive side of the anode voltage to the cathode.

To test bipolar transistors, the screen grid voltage is increased to generate a base current. This current is limited to 4 mA at 400 V by R2 and R3. To test MOSFETs, R1 is switched in to form part of a potential divider that generates the gate voltage. D3 and D4 limit V_{gs} to a maximum of 19 V.

Zener diodes and $V_{ce\ max}$ for bipolar transistors

If the 'Zener diode' option is selected then the text in the the 'Uce max' and 'Ic max' fields changes to 'Uz' and 'Pmax' respectively (see **Figure 16**) as, in the case of Zener diodes, it is more usual to quote a maximum power.

Upon pressing 'Start curve' the anode voltage is swept from zero up to either U_z or until

Figure 14. Adaptor for a type EL34 valve.

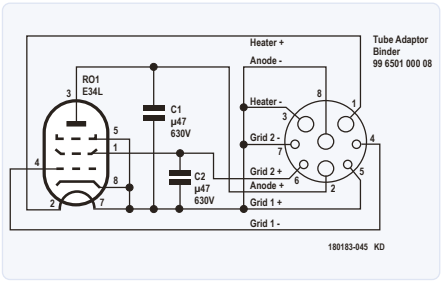


Figure 15. Circuit diagram of the valve adaptor.

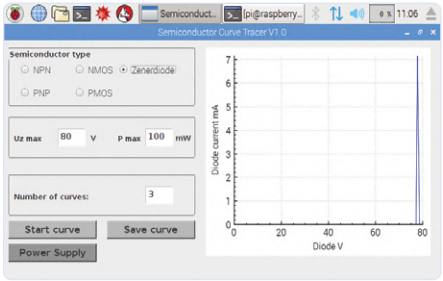


Figure 16. The maximum collector-emitter voltage of a sample BC547C.

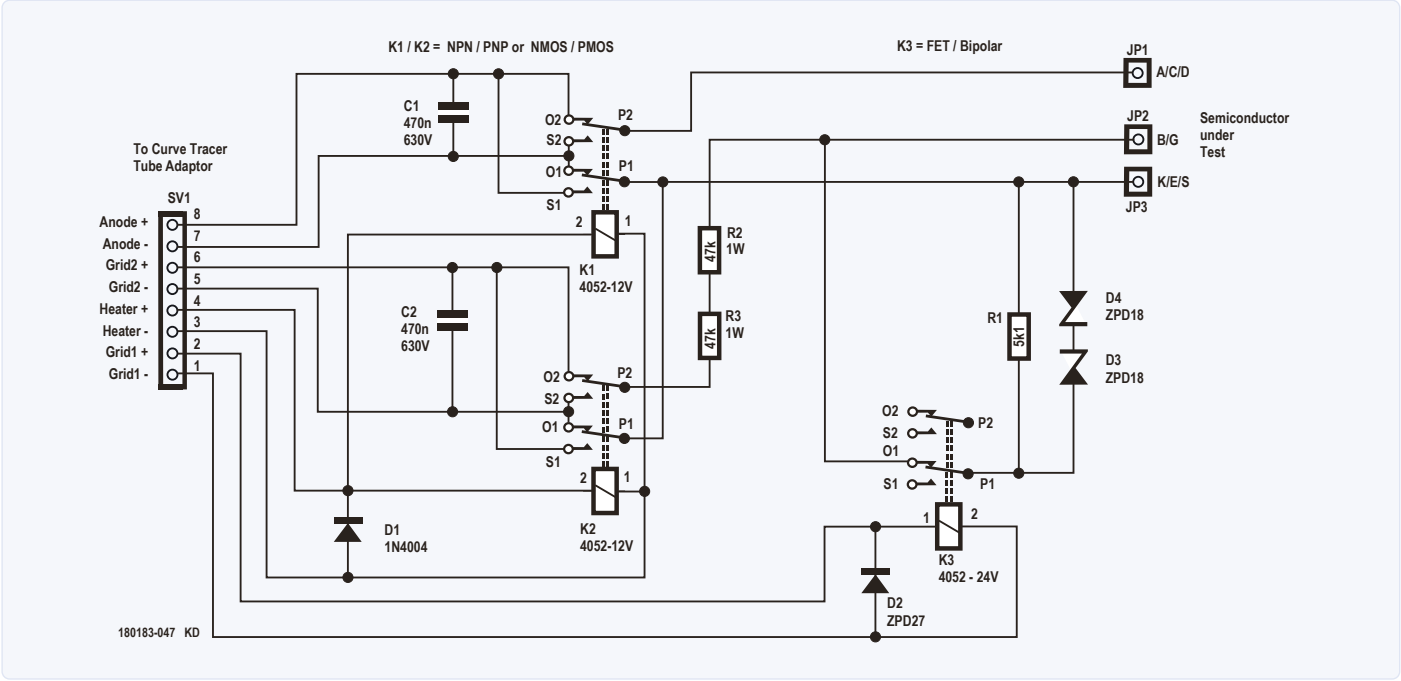


Figure 17. Circuit diagram of the semiconductor adaptor.

P_{max} is reached, whichever occurs first. If the current through the diode at U_z is less than 1% of the maximum Zener current, then the error message shown in **Figure 20** is displayed. Either the Zener diode is defective, or U_z needs to be increased.

Because of the high voltage capability of the anode supply it is also possible to measure the maximum collector-emitter voltage with open base ($V_{ceo\ max}$) of many bipolar transistors. The curve in **Figure 16** is the result of testing a BC547C. Although the datasheet suggests a limit of only 45 V, this particular sample sustains a V_{ceo} of over 75 V.

Testing bipolar transistors

When testing a bipolar transistor it is necessary first to select between NPN and PNP.

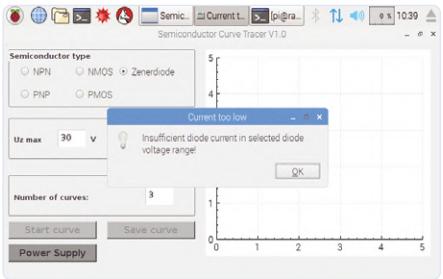


Figure 20. Error message reporting that the Zener diode current is too low.

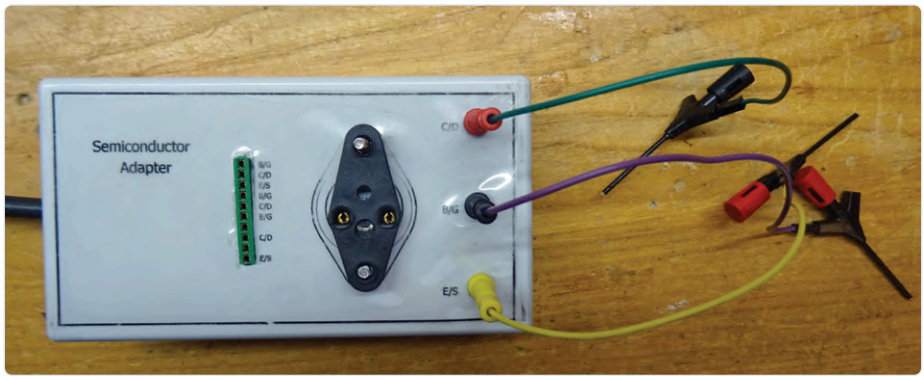


Figure 18. Outside view of the semiconductor adaptor.

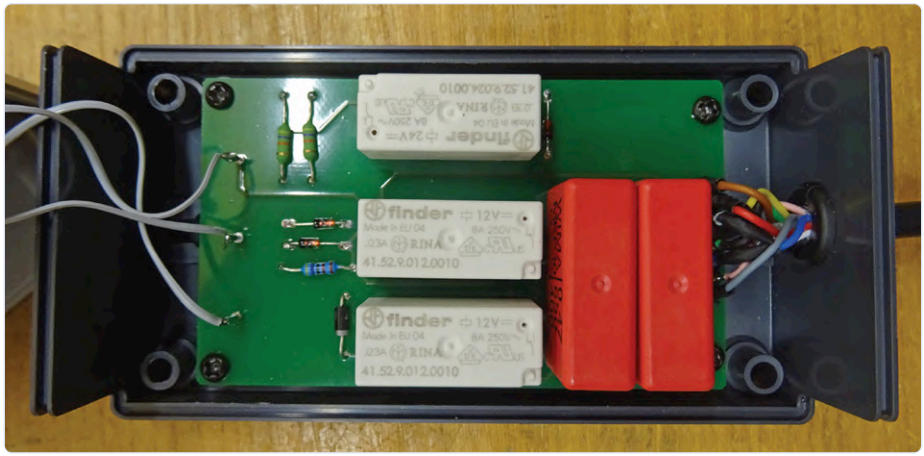


Figure 19. Inside view of the semiconductor adaptor.

Figure 21. Front view of the prototype supply.

Values for $V_{ce\ max}$ and $I_{c\ max}$ should also be set, as well as the desired number of curves to be traced. First, the software checks whether any significant collector current flows in the absence of a base current. If so, the test stops with an error message. Otherwise, the collector voltage is set to the given value and the base current is increased until $I_{c\ max}$ is reached. Again, if the maximum collector current setting cannot be reached with a maximum base current of 4 mA, an error message is displayed.
The characteristic curves $I_c = f(V_{ce})$ will then be displayed with the corresponding base currents in the key. The 'Save curve' button lets you store the raw I_c, I_b and V_{ce} data in a CSV-format file.

Testing MOSFETs

When testing a MOSFET it is necessary to first select between NMOS and PMOS. Values for $V_{ds\ max}$ and $I_{d\ max}$ should also be set, as well as the desired number of curves to be traced. First, the software checks whether any drain current flows with $V_{gs} = 0$ and, if so, the test stops with an error message. Otherwise, the drain-source voltage is set to the given value and the gate voltage increased until $I_{d\ max}$ is reached. Again, if the maximum drain current setting cannot be reached with the maximum gate voltage of 18 V an error message is displayed.
The characteristic curves $I_d = f(V_{ds})$ will then be displayed with the corresponding gate voltages in the key. The 'Save curve' button lets you store the raw I_d, V_{gs} and V_{ds} data in a CSV-format file.

Caution!

Take care when setting the maximum drain or collector currents as well as the maximum drain-source or collector-emitter voltages. At 400 V and a current of 300 mA the transistor under test will be dissipating 120 W!

Pay attention to the SOA of the device as given in its datasheet; and you may need to equip the transistor under test with a heatsink.

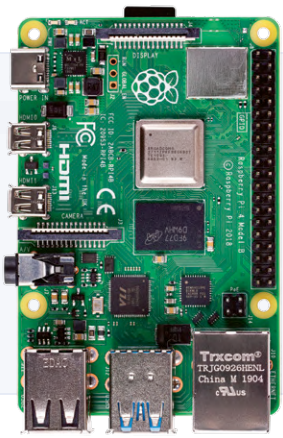
Finally

The printed circuit board design files (in Eagle format) and parts lists are included in the free archive download [1] along with enclosure drawings, LTSpice files, and the complete source code, all freely available for private use. **Figure 21** shows the final prototype.

180183-03

SHOPPING LIST

- **Raspberry Pi 4 B (2 GB RAM)**
  www.elektor.com/raspberry-pi-4-b-2-gb-ram
- **7" touchscreen for Raspberry Pi**
  www.elektor.com/joy-it-7-touchscreen-for-raspberry-pi
- **JOY-iT JT-RD6006 DC power supply bundle**
  www.elektor.com/joy-it-jt-rd6006-dc-power-supply-bundle



WEB LINKS

[1] **Download accompanying this article**: www.elektormagazine.com/180183-03
[2] **FrontDesign**: www.schaeffer-ag.de/en/front-panel-designer
[3] **Visual Studio**: https://visualstudio.microsoft.com/downloads/
[4] **Cross compiler for Raspberry Pi**: https://visualgdb.com/download/
[5] **Qt Designer**: https://build-system.fman.io/qt-designer-download
[6] **WiringPi**: http://wiringpi.com/
[7] **WinSCP**: https://winscp.net/eng/download.php
[8] **QCustomPlot**: www.qcustomplot.com/index.php/download
[9] **Matchbox keyboard**: https://github.com/xlab/matchbox-keyboard

Hexadoku The Original Elektorized Sudoku

Traditionally, the last page of *Elektor magazine* is reserved for our puzzle with an electronics slant: welcome to Hexadoku! Find the solution in the gray boxes, submit it to us by email, and you automatically enter the prize draw for one of five Elektor book vouchers.

The Hexadoku puzzle employs numbers in the hexadecimal range 0 through F. In the diagram composed of 16 × 16 boxes, enter numbers such that **all** hexadecimal numbers 0 through F (that's 0-9 and A-F) occur once only in each row, once in each column and in each of the 4×4 boxes (marked by the thicker black lines).

A number of clues are given in the puzzle and these determine the start situation.

Correct entries received enter a prize draw. All you need to do is send us **the numbers in the gray boxes**.

SOLVE HEXADOKU AND WIN!

Correct solutions received from the entire Elektor readership automatically enter a prize draw for five Elektor Book Vouchers worth **€50.00 each**, which should encourage all Elektor readers to participate.

PARTICIPATE!

Ultimately October 5, 2020, supply your name, street address and the solution (the numbers in the gray boxes) by email to: **hexadoku@elektor.com**

PRIZE WINNERS

The solution of Hexadoku in edition 4/2020 (July & August) is: **8C5B2**.
The book vouchers have been awarded to: Heinz Seitz (Germany), Kathryn Rangeley (United Kingdom), Mihails Sehurins (Latvia), Martin Poelstra (The Netherlands), and Eric Poole (Australia).

Congratulations everyone!

		7	6	8	4							9			3
E				3	6	F	7		B		2	5	0	4	
					A			F		D			B		6
B			5	1	2	C		3					7	A	8
	3				5	2			1	0			F	7	
						1	3	9	F		8			C	
1			E				F		D	2			5		
	C						4			5		B			
	1						8			6		A			
D			9				0		2	4			1		
						7	9	D	5		A			8	
	6				1	A			8	E			9	D	
2			C	7	F	B		0					A	1	5
					E			6		7			D		C
4				A	0	3	2		C		D	8	6	B	
		A	F	5	C							7			0

6	D	7	A	B	9	3	5	C	E	4	0	2	F	8	1
F	9	0	5	4	7	C	1	8	A	D	2	6	B	E	3
4	1	C	2	6	E	8	D	7	F	3	B	5	9	A	0
3	B	E	8	F	0	A	2	9	1	5	6	7	C	D	4
0	A	D	1	7	B	4	3	E	9	C	5	8	6	F	2
B	6	2	9	5	F	0	C	A	7	1	8	E	3	4	D
C	5	3	4	1	6	E	8	B	D	2	F	0	7	9	A
E	F	8	7	9	D	2	A	0	3	6	4	B	1	C	5
7	4	6	0	C	1	5	E	D	B	8	3	F	A	2	9
A	3	9	F	D	4	B	7	2	0	E	1	C	5	6	8
8	C	5	B	2	3	6	0	F	4	9	A	D	E	1	7
2	E	1	D	8	A	F	9	5	6	7	C	3	4	0	B
D	2	A	6	0	5	7	F	1	C	B	9	4	8	3	E
1	0	B	C	A	8	D	4	3	5	F	E	9	2	7	6
9	8	4	E	3	C	1	B	6	2	0	7	A	D	5	F
5	7	F	3	E	2	9	6	4	8	A	D	1	0	B	C

The competition is not open to employees of Elektor International Media, its subsidiaries, licensees and/or associated publishing houses.

Analogue Electronics Design

Case Study #2 — Part 1: Analogue filter theory

By **Alfred Rosenkränzer** (Germany)

This series of three articles deals with the design of analogue filters. The first part covers the basic theory with an emphasis on practical applications. The second deals with the design of active filters, while the third and final part will cover passive filters.

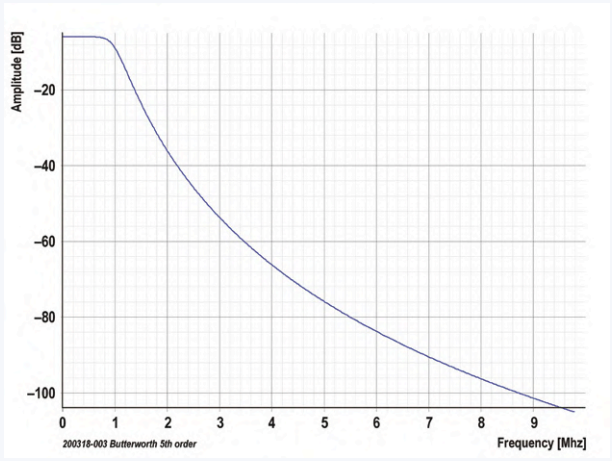


Figure 1: Frequency characteristic of a 1MHz low-pass filter; linear frequency, amplitude in dB.

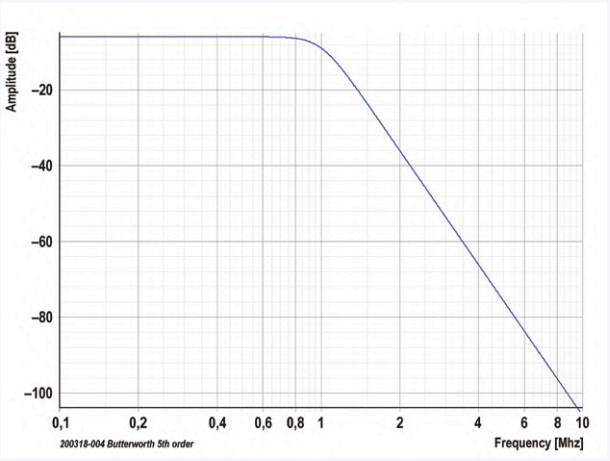


Figure 2: Frequency characteristic in the audio range; logarithmic frequency, amplitude in dB.

Analogue filters are built using resistors, inductors, capacitors, and opamps, and are used to process analogue voltages and currents. Digital filters, in comparison, are based on digital circuits such as flip-flops, counters, multipliers (usually in the form of a DSP or FPGA) or software in microcontrollers and SoCs, and operate on streams of data.

Filter characteristics

The relevant characteristics of a filter are generally represented on a graph. The most important of these represents amplitude as a function of frequency, i.e. the filter's frequency characteristic. The ratio between the input and output voltages is rarely expressed as a linear (in percent) value along the Y axis, but more usually as a logarithmic quantity in dB. The frequency is represented along the X-axis typically using a linear scale for narrower bandwidths (**Figure 1**). Audio ranges, however, are often shown on a logarithmic scale (**Figure 2**). All graphs in this article were made using the Simetrix simulation software [1] but have been post-processed for reasons related to the technical details of reproduction.

A filter does not only change the amplitude of a signal; it also changes its phase. When

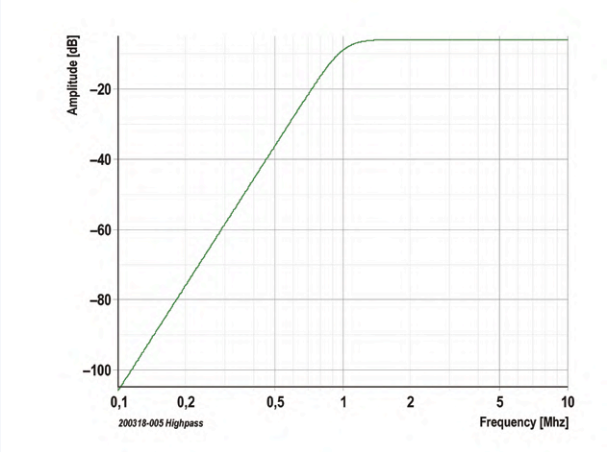


Figure 3: Frequency characteristic of a 1MHz high-pass filter; log-log scale.

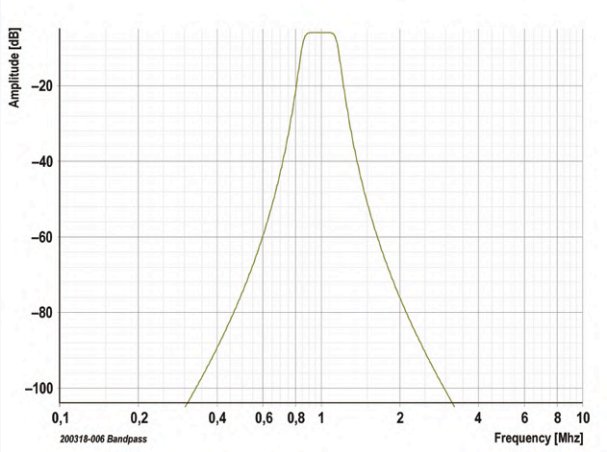


Figure 4: Frequency characteristic of a 1MHz band-pass filter; log-log scale.

viewing a filter's phase characteristic, the phase shift is expressed in degrees along the Y-axis. The frequency is, just as before, represented along the X-axis. I don't find the phase characteristic particularly informative, certainly not in the RF ranges. Of much more interest is the group delay as a function of frequency. This corresponds to the derivative of the phase characteristic with respect to frequency. With this you can easily see how long it takes for a signal of a particular frequency to pass through the filter. If the group delay is flat, then all frequencies require the same amount of time. The importance of this will be made clear with a few diagrams a little further on. The step response of a filter is important in the time domain. It shows the linear amplitude curve of the output signal as a function of time when a step signal is applied to the input. These characteristics will also be looked at later.

Filter types

A **low-pass filter** allows low frequencies to pass mostly unhindered and attenuates the high frequencies. In between is a transition region. The point at which the signal is attenuated by 3 dB is the border between the pass band and the transition region. The corresponding frequency is therefore an important filter parameter. After a certain amount of attenuation the stop band is reached. There are no generally accepted numbers for this because it depends on the specific application. **Figures 1** and **2** show the frequency characteristic of low-pass filters. A low-pass filter will also allow DC to pass.

A **high-pass filter** allows the high frequencies to pass and attenuates the low frequencies. Here the -3 dB point designates the move from transition region to pass band (**Figure 3**). The comments for the low-pass filter are also valid for the high-pass filter. However, a high-pass filter will block any DC component of the input signal.

A **band-pass filter** allows a certain frequency range to pass and attenuates both the lower and higher frequencies. There are two -3 dB points that define the pass band (**Figure 4**). DC voltages are blocked because of the high-pass section.

The opposite of a band-pass filter is the **band-stop filter**. Here a certain frequency range is attenuated while the lower and higher frequencies are allowed through (**Figure 5**). The low-pass section allows DC signals to pass through.

An **all-pass filter** does not change the amplitude: it is a filter with a flat frequency characteristic. However, the group delay and phase (**Figure 6**) of the signal do change as a function of frequency. If you add the output signals of (multiple) all-pass filters to the original signal, the result is a comb filter with corresponding dips (*notches*) in the amplitude characteristic.

Filter configurations

So, how do you design a filter and determine its characteristics? This is, certainly for filters of higher order, definitely not trivial. Mathematicians of repute have already developed mathematical solutions according to special rules calculated many years ago. These rules are still relevant today, although all the labour of calculating these has now been taken out of our hands by a PC and special filter design software.

To honour these mathematicians, their filter types are named after them. Several common filter types (without claim to completeness) are:

- Bessel
- Butterworth
- Chebyshev
- Cauer

In addition, many further filter variants can be conceived and constructed – you can even combine different characteristics. We will take a look at the most important characteristics of these filter types to simplify the selection for a particular application. **Figure 7** shows the four different passive, low-pass filters. The corresponding frequency characteristics are illustrated in **Figure 8**.

Although the schematics for the three top-most filters are identical, apart from their component values, there are clear differences in the frequency characteristics. With the Bessel filter the transition from the pass-band to the stop-band is very smooth. A Butterworth filter has a much steeper characteristic. A Chebyshev filter is steeper again. The Cauer filter, with two additional capacitors in parallel with the inductors, realises an even steeper transition to the stop band.

The frequency characteristic is therefore not monotonic but, especially at higher frequencies, its damping remains virtually constant and smaller than for the other three filter types. The locations of the 'dips' depend on the component values. When choosing appropriate values, these notches can be used to suppress specific frequen-

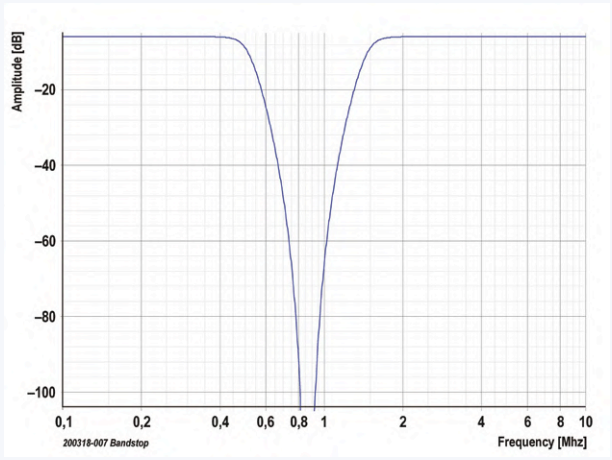


Figure 5: Frequency characteristic of a band-stop filter; log-log scale.

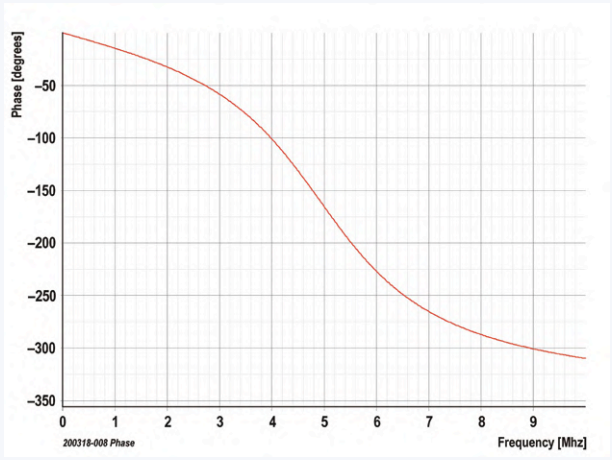


Figure 6: Frequency characteristic of an all-pass filter; linear frequency.

cies. Depending on the design, the 'bump' between the two dips and the attenuation at higher frequencies can be changed.
In **Figure 9** you can see the ripple in the pass band and the amplitude characteristic around the -3 dB point in more detail. The Bessel filter starts to attenuate quite early in the pass band, but very gradually. The behaviour of the Butterworth filter is nearly ideal, comparable to a simple RC low-pass filter, but its steepness is not better than average. The Chebyshev and Cauer filters display a characteristic ripple in the pass band with an amplitude of 1 dB. These filters only have an attenuation of -1 dB at the nominal frequency, rather than 3 dB. This needs to be taken into account when designing the filter. And, for the Chebyshev and Cauer filters, the following is also true: the steeper the filter, the more ripple there is in the pass band.

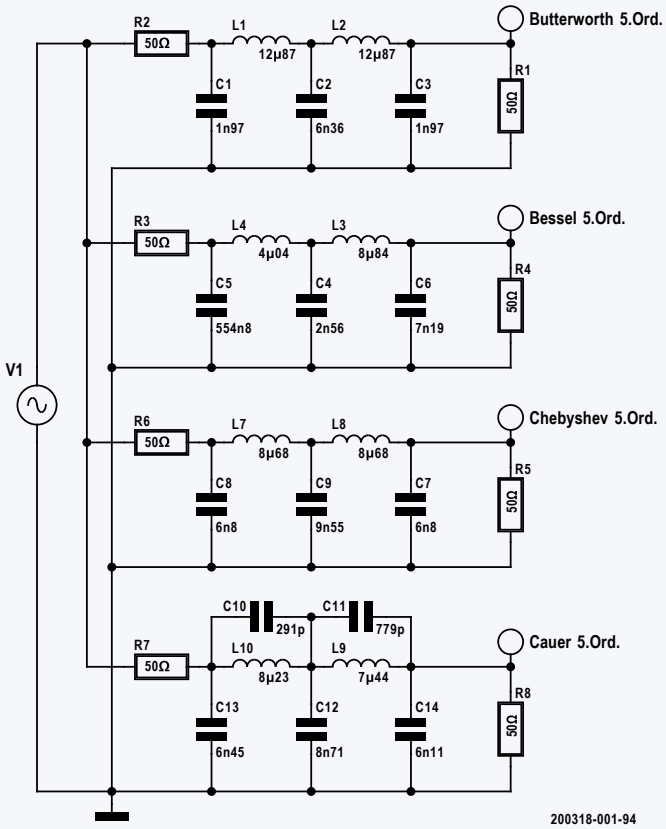


Figure 7: Passive 5th-order 1MHz low-pass filters (Butterworth, Bessel, Chebyshev, Cauer).

Figure 10 shows that the group delay of a Bessel filter is independent of the frequency. With a Butterworth filter, the group delay increases monotonically with frequency. With the Chebyshev and Cauer filters, the group delay shows a certain amount of ripple.
The actual characteristic of the output signals is also interesting when the filters have to deal with steep edges (when they are driven by a square wave, see **Figure 11**). The step response of a Bessel filter (**Figure 12**) shows practically no overshoot. The Butterworth filter (blue curve) shows a clear but quickly diminishing overshoot. With the Chebyshev and Cauer filters we see something like a damped oscillation. The signal edges of the four filters don't align either, which is to be expected considering their different group delays.
The ripple on the signal can be viewed in the 'spectral' domain as well: according to Fourier, a square wave can be considered as the sum of its sinusoidal frequency components. These frequencies, when passing through the filter, experience different amplitude changes and potentially also different shifts in time. It is, therefore, not surprising that we don't see a monotonic

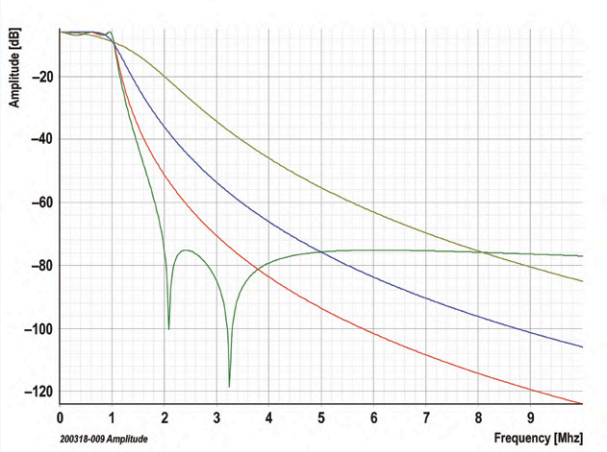


Figure 8: Frequency characteristics of the four low-pass filters of Figure 7 (logarithmic amplitude, linear frequency). Light green = Bessel, blue = Butterworth, red = Chebyshev, and dark green = Cauer.

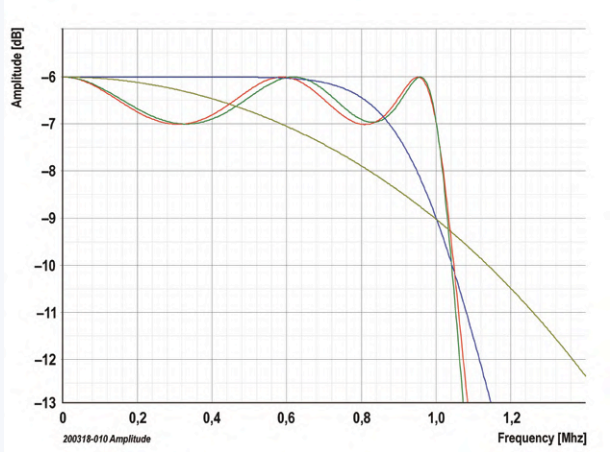


Figure 9: Enlargement of a section of Figure 8. Here you can see the characteristic in the pass band and around the -3 dB point more clearly.

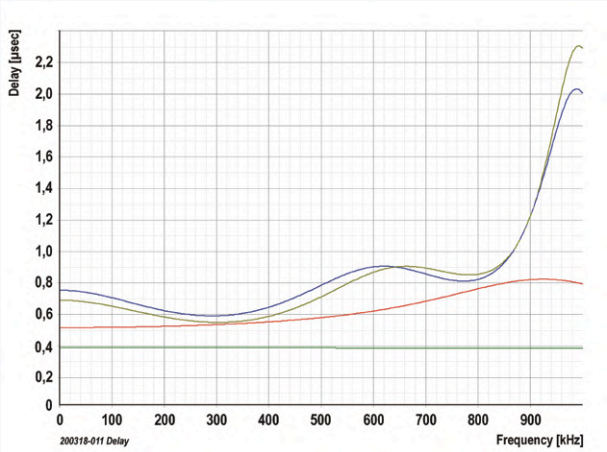


Figure 10: Group delay of the four filters in the pass band as a function of the frequency. Dark green = Bessel, red = Butterworth, light green = Chebyshev and blue = Caue.

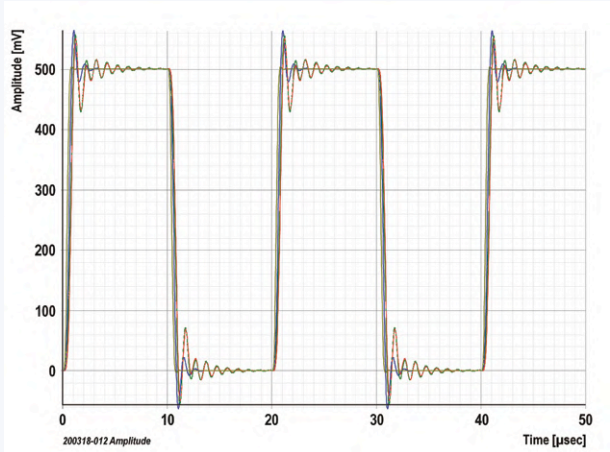


Figure 11: Characteristics of the output signals of the four filters when a square wave is applied. Light green = Bessel, blue = Butterworth, red = Chebyshev and dark green = Cauer.

curve at the output of these filters. While the higher frequency components are indeed attenuated in the Bessel filter, they are not displaced with respect to the lower frequencies because of its constant group delay. This effectively increases the rise and fall times and rounds the corners of the square wave. The Chebyshev and Cauer filters, by comparison, show a clearly visible overshoot and oscillation after a rapid change of the input signal (signal edge), a consequence of their strongly frequency-dependent group delay.

In the past, this behaviour was very disturbing in analogue video applications. With the addition of all-pass filters, the group delay was flattened to reduce the overshoot and centre it around the signal edges. The design of such all-pass filters for the correction of the group delay is a science in itself. Only a few design programs are capable of carrying out these calculations. Furthermore, a great deal of additional components are required compared to a 'pure' low-pass filter.

Criteria for filter selection

In **Table 1** we have summarised the most important characteristics of the four most common filter types for you. Here we repeat, once more, their salient features:

- A Bessel filter exhibits optimal impulse behaviour and is, therefore, suited to turn a steep digital signal into a pulse with 'rounded corners'. Because of its limited roll-off, this type of filter is seldom used for frequency suppression.
- The Butterworth-filter offers a good compromise between filter roll-off (steepness) and impulse behaviour. The amplitude characteristic shows no ripple.
- Chebyshev and Cauer filters have a steep slope at the cut-off frequency, but this comes at the expense of ripple in the pass band, as well as pronounced and protracted oscillation artefacts in the time domain.

Table 1. Filter characteristics

Filter type	Pass band	Stop band	Stop band	Steepness
	behaviour	behaviour	damping	
Bessel	slowly reducing	slowly reducing	small	small
Butterworth	flat	increasing attenuation	average	average
Chebyshev	with ripple	increasing attenuation	large	large
Cauer	with ripple	with ripple	large	large

Table 2. Terminology

Technology	Analogue passive Analogue active Digital
Type	Low pass Band pass Band stop All pass
Configuration	Bessel Butterworth Chebyshev Inverse Chebyshev Cauer Lehmann Legendre Gauss Raised cosine TBT
Impedance	Input impedance Output impedance
Frequency	Cut-off frequency Stop frequency Pass band Stop band Bandwidth
Damping	Pass band damping Stop band damping
Miscellaneous	Group delay Impulse behaviour Order Complexity Sensitivity to component tolerances

What remains is to examine how the characteristics of a filter change with its order (or, expressed differently, the number of frequency-determining components). As an example, we have drawn Butterworth filters of 5th, 6th, 7th, and 9th order in **Figure 13**. When you look at the component values, you will see a symmetry in their values for the odd-numbered orders. In **Figure 14** you can see clearly how the steepness of the filter increases with order. **Figure 15** shows an enlarged view of what happens around the -3 dB point: the characteristics of all the filters pass exactly through this point. However, filters of higher orders begin to attenuate at ever higher frequencies and then roll off much more steeply.

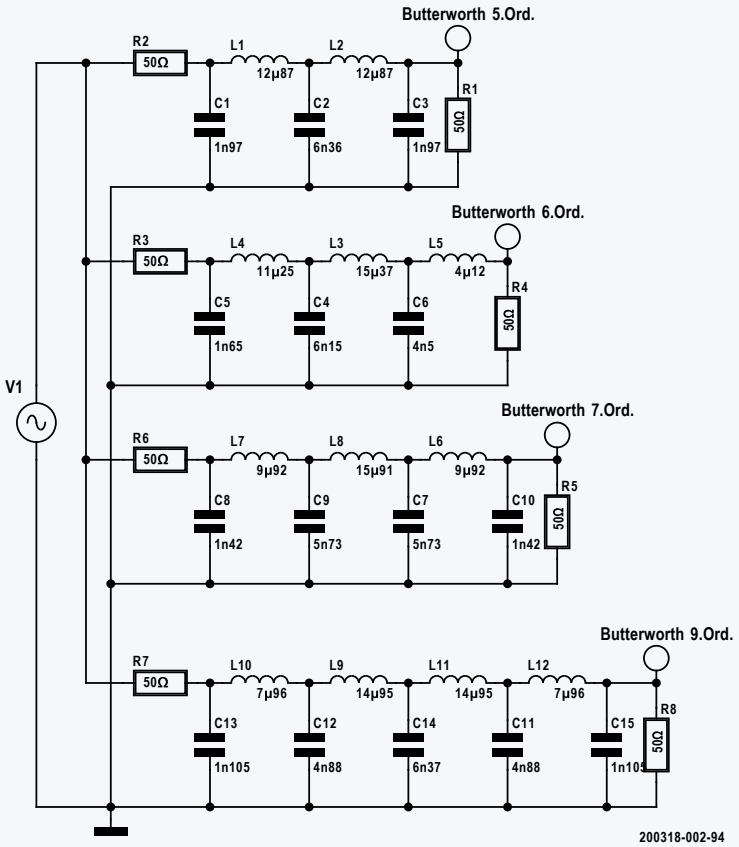


Figure 13: Butterworth filters of 5th, 6th, 7th,and 9th order.

The higher the order, the greater the group delay (**Figure 16**) and the overshoot/oscillatory behaviour (**Figure 17**), both in amplitude and duration.

So, the steepness of a filter is determined by the chosen configuration (the filter type) as

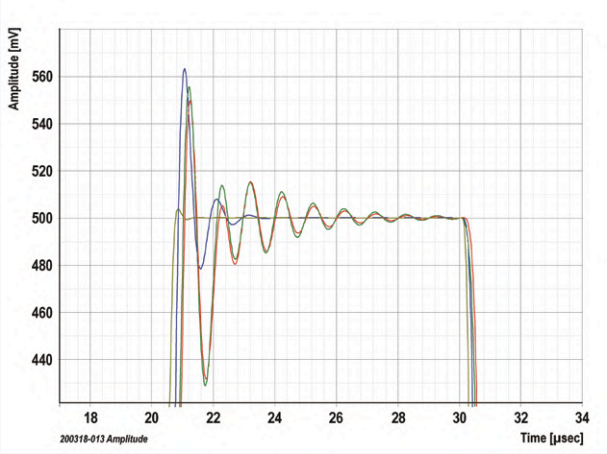


Figure 12: The step response is clearly visible here (enlargement of Figure 11). Light green = Bessel, blue = Butterworth, red = Chebyshev and dark green = Cauer.

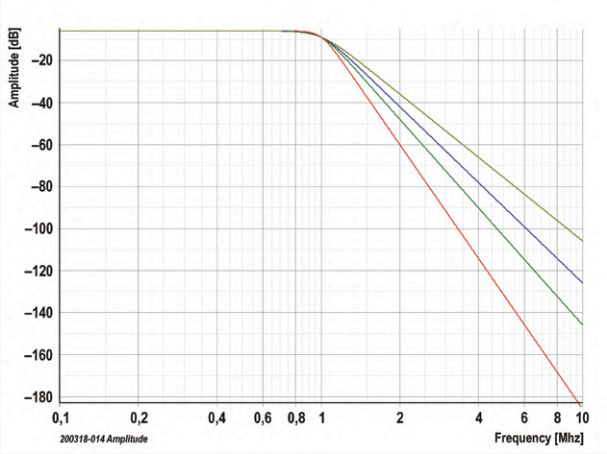


Figure 14: Frequency characteristics of the filters of Figure 13; log-log scale. 5th order = light green, 6th order = blue, 7th order = dark green, and 9th order = red.

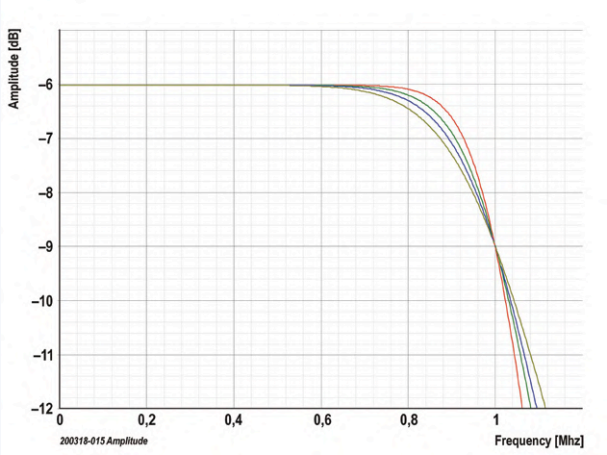


Figure 15: The same characteristics as in Figure 14, enlarged around the -3 dB point.

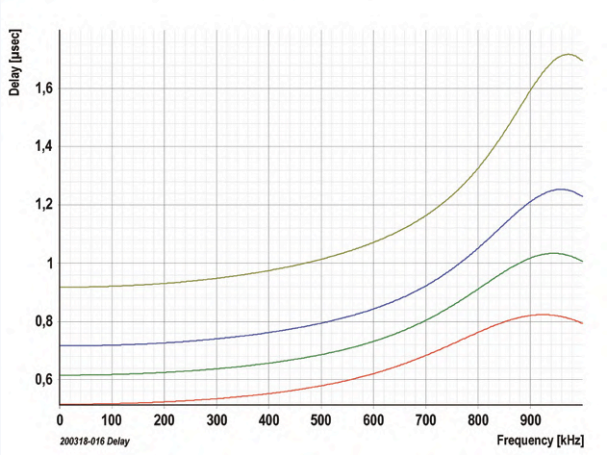


Figure 16: Group delays as a function of frequency for the four filter types. Light green = 9, blue = 7, dark green = 6 and red = 5.

well as the order. Thus, by selecting a higher order, and therefore also a larger number of components, even a steep Bessel filter can be constructed.

In **Table 2** we have listed the most important terms that you are likely to encounter in the world of filters.

200318-04

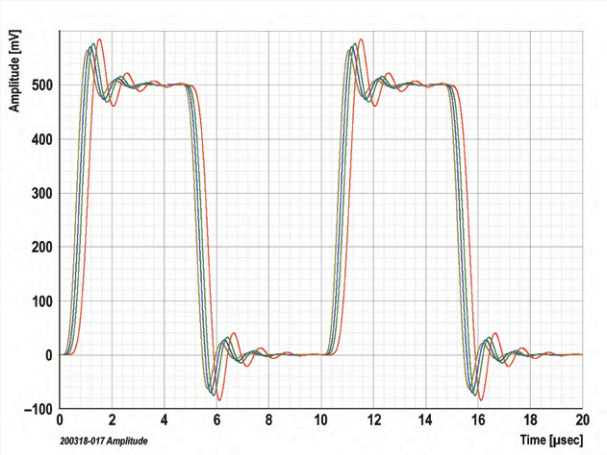


Figure 17: Step response with four different filter types. Light green = 5, blue = 6, dark green = 7 and red = 9.

WEBLINK

[1] Simetrix: http://www.simetrix.co.uk

Rigol DP832 3-ch Programmable DC Power Supply (0-30 V, 0-3 A, 195 W)

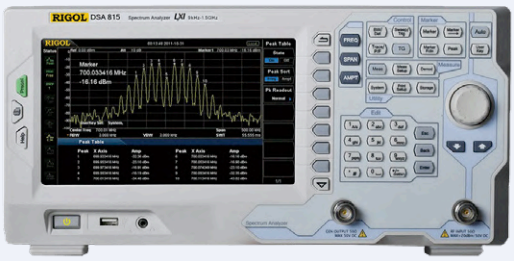
was ~~€549.00~~
now €449.00

www.elektor.com/17823

Rigol DG2052 Function/ Arbitrary Waveform Generator (50 MHz)

was ~~€699.00~~
now €579.00

www.elektor.com/19345



Rigol DSA815-TG Spectrum Analyzer (9 kHz to 1.5 GHz)

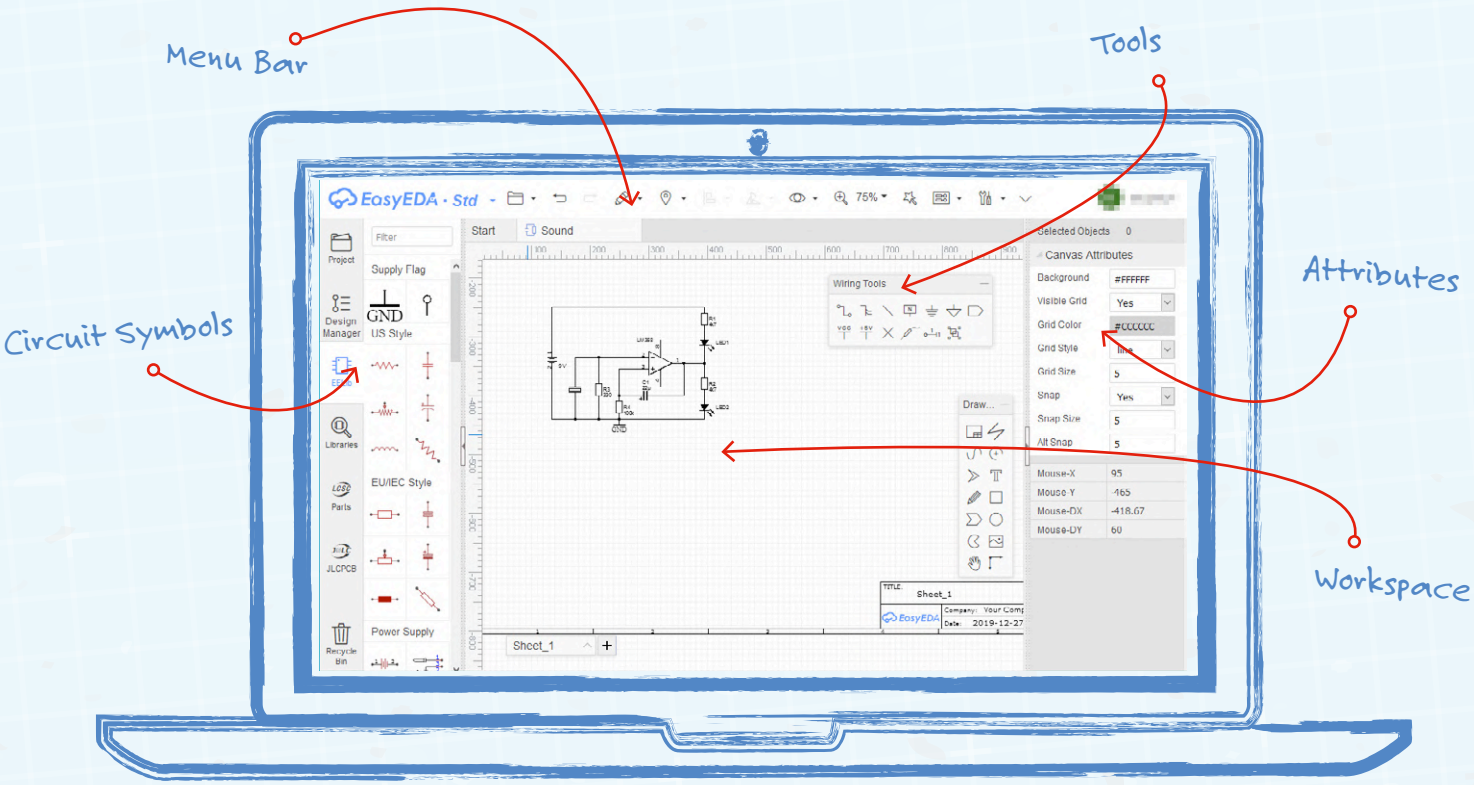
was ~~€1,549.00~~
now €1,299.00

www.elektor.com/19349

Rigol DG2102 Function/ Arbitrary Waveform Generator (100 MHz)

was ~~€1,149.00~~
now €949.00

www.elektor.com/19347



How to Draw a Circuit Diagram

By **Florian Schäffer** (Germany)

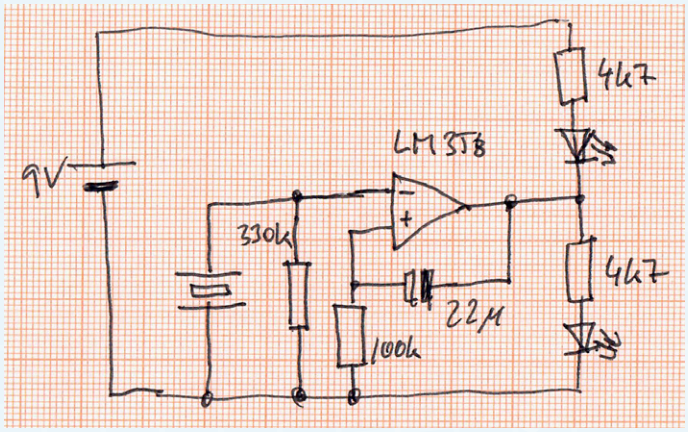
When you have an idea for a circuit and want to share it with others, or even just document it for yourself, it is time to draw a circuit diagram. Using EasyEDA is almost as quick as drawing by hand, and the final appearance is decidedly more attractive.

In theory, it is possible to draw circuit diagrams with any drawing package. However, on the downside, you will have to create all the symbols yourself and you will not benefit from the special features that a dedicated EDA (electronic design automation) package can offer. Such design software not only simplifies the development of the circuit diagram by providing libraries containing symbols for a wide range of components; they can also offer features that check and track down errors in the design or, in some cases, simulate your circuit. They can also be used to convert your circuit diagram into a PCB (printed circuit board) layout.

Which EDA software is best?

Of course, there is no definitive answer to this question. The advantage of EasyEDA, as described here, is that it is free and works in the browser without the need to install software locally. For occasional use and for beginners it is practical and simple to use. Fritzing is also capable of creating circuit diagrams and can generate breadboard wiring diagrams too. However, the results are rather unsightly and, more importantly, the resultant schematic does not conform to international standards. KiCad and gEDA are freely available but they have a steep learning curve and initial installation can be challenging. Target 3001! is a low-cost commercial program that is also offered in a (severely restricted) free version for private users. In the upper echelons there are programs such as Altium Designer and EAGLE. The latter is also available as a free version, but with feature limitations.

In this brief workshop we will look at the basic operation of EasyEDA. Once you have your circuit diagram ready, this website will also let you design your PCB layout, but this is an aspect we will not be covering here.

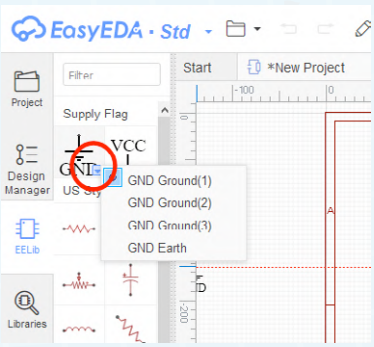


A sketch of a simple circuit that we want to enter into the EDA software. The precise function of this circuit is not relevant to this article.

1. Sign in to the online editor website at **https://easyeda.com/editor** using the 'Login' button at the top right. The first time you use the system you may wish to create an account using 'Register.' You do not need to register just to draw circuit diagrams, but registration is required if you want to save or export them.

2. You can change the interface language if you wish. Some language versions have incomplete or inaccurate translations, but the English version works well. Start a new sheet with *Document / New / Schematic*.

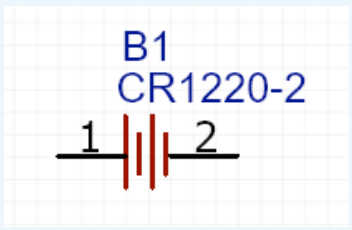
3. Down the left-hand side you will find various options. Start by selecting *EELib*, which will bring up the most commonly-used component symbols in the next column. You can scroll through the list using the mouse wheel. Symbols are available both in the US style and in the EU style according to IEC (International Electrotechnical Commission) standards. When you hover with the mouse over a symbol, a small triangle will usually appear. Clicking on this triangle brings up a context menu that lets you choose particular package sizes or types. These can change the appearance of the symbol or affect the footprint used when laying out a circuit board. In most cases you will not need to use this feature if all you are doing is drawing a circuit diagram.



4. In the area labelled *Power Supply* select the battery symbol by clicking on it. EasyEDA will only offer you a choice of button cells but, since the footprint of the component is not important to us for the moment, this does not matter.

5. The chosen symbol will now move around the screen with the mouse pointer. The mouse wheel can be used to zoom in and out on the working area.

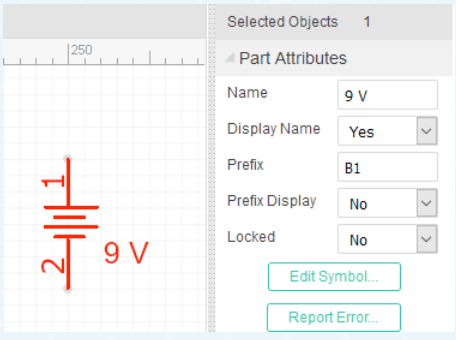
6. Click in the working area in order to place a copy of the component symbol. Further clicks will add more copies of the same symbol, which in this case we do not need. You can leave this mode by pressing the Escape key.



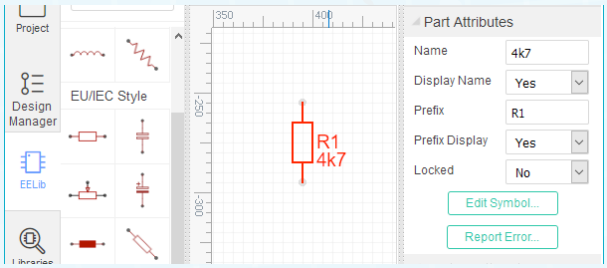
7. You can move a component symbol at any time using the mouse, or delete it by pressing the Delete key. Take care to click on the component symbol itself and not on its annotation.

8. Select the battery symbol by clicking on it, then rotate it using the *Rotate Right* option in the *Rotate and Flip* menu.

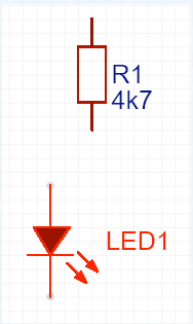
9. On the right-hand side of the window you can change various attributes for the component symbol currently selected. Change the *Name* entry to '9 V' and under *Display Prefix* select 'No.'



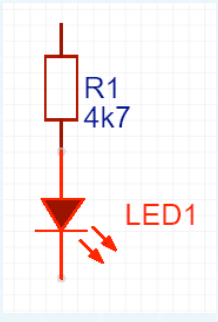
10. Add a resistor, rotating it right through 90°, and change its name in the attribute area to the value '4k7.'



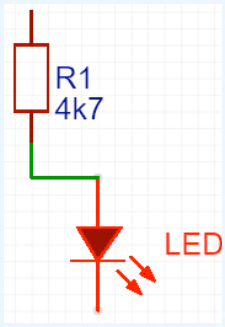
11. Add the symbol for an LED (light-emitting diode) to the circuit diagram near to the resistor you have just placed. Rotate the LED clockwise and, in the attribute area, disable the display of the component name by setting *Display Name* to 'No.'



12. Now we need to connect these two components together electrically. To do this we create a wire by simply bringing the two components together so that the desired connections touch.

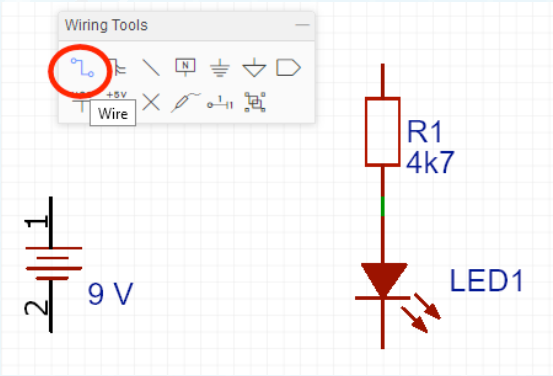


13. EasyEDA has now created an (invisible) connection. If you drag the symbols apart a little you will see that a green connecting wire has been created.

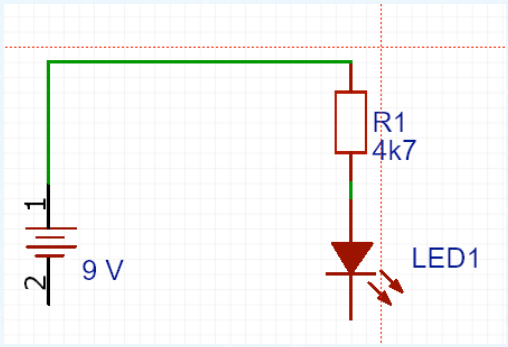


14. You can also edit this wire by clicking on it. This allows you to then move it around or even delete it. As long as the connection exists, it will be maintained when you move the symbols. This is one of the great advantages of using dedicated EDA software.

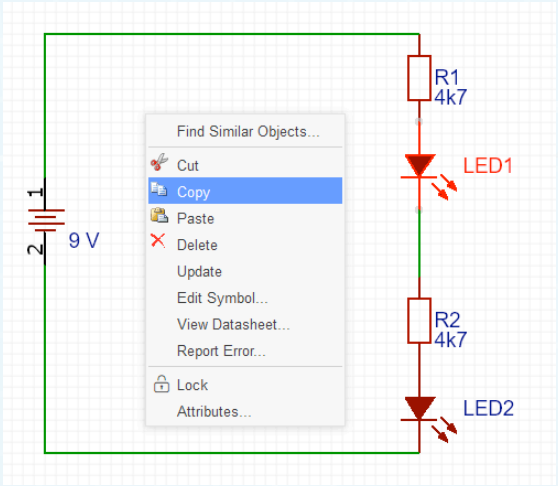
15. You can also create connections manually, for example to connect the other terminal of the resistor to the battery. You should be able to see two toolbox windows (if not, use *View / Wiring Tools*). It is essential that electrical connections are only created using *Wire* and not with a *Line* from the *Drawing Tools* toolbox. This latter option is only used for drawing lines to, for example, make a frame around your schematic.



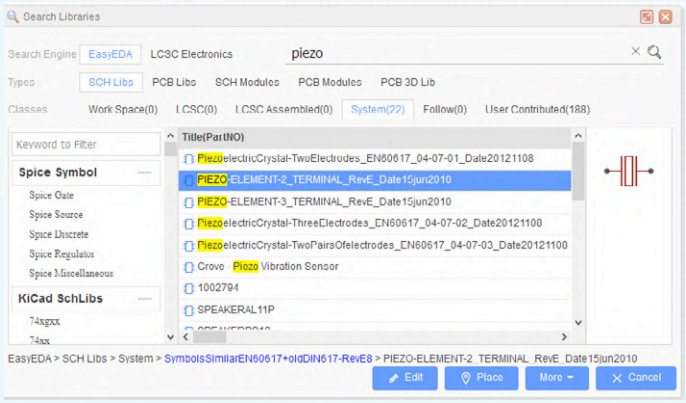
16. Select *Wire* and click on the battery terminal. A green line will then appear from that point running to the current mouse position. You can now extend the wire directly to the unconnected end of the resistor with the software choosing the route of the wire. If you are not happy with its choice of route, you can click on it and add new intermediate points. You end editing of the wire by clicking on a different wire or on a component terminal.



17. Add the second LED and the second resistor and connect the components to one another and to the battery. You can do this either by creating new symbols from the selection on the left, or by using the clipboard in the conventional way with *Copy* and *Paste*. This is accessible from the *Edit* menu or from the context menu that pops up when you right-click with the mouse.



18. Two of the component symbols that we need are not available in the standard list: the piezo transducer and the operational amplifier. For the latter, click at the far left on *Library*, which will open a window to let you search for components. In the search box you can enter either a part number or a description. Enter 'LM358' here and then click on the magnifying glass symbol immediately to the right.



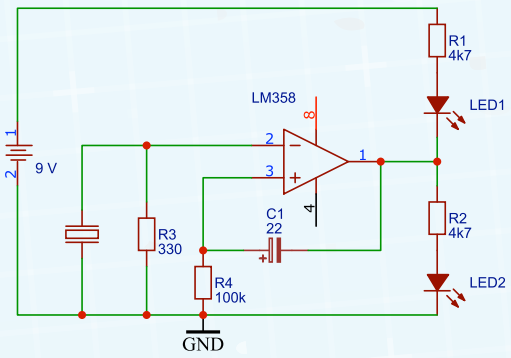
19. After a moment the search results will appear. There are multiple entries that differ in the design of the component symbol (in the right-hand pane when you select a row) and in the physical footprint of the actual part. For our example, the best choice is the LM358APWR. This device contains two operational amplifiers (we are not, for the moment, concerned about what an operational amplifier is or how it works) in a single package. Choose the second sub-option (*LM358APWR.2*) and click on *Place*. The window will close and you can place the symbol in the working area as before.

20. You will probably need to move the existing components around a little to make space for the new ones. We will ignore the power supply pins (4 and 8) of the operational amplifier for now.

21. A search is also required to find the piezo transducer. Again, click on *Library* and enter the search term 'piezo.' The second result in the list looks correct. Click on *Place* and add it to the circuit diagram.

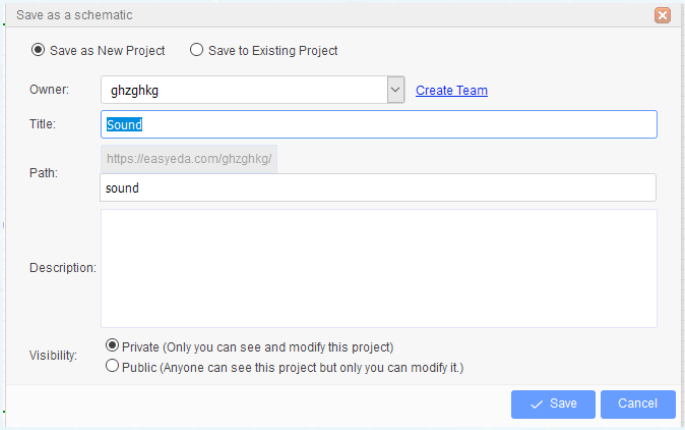
22. Adding the remaining components and their connections should not present any difficulties. EasyEDA respects the rules for adding nodes where wires join. If two wires simply cross, no dot is added. However, if you click on a wire while laying down another wire, a connection is formed and a dot, to indicate a connection, is added.

23. For the sake of completeness we should add a ground symbol. Choose this from the area labelled *Supply Flag* by opening the context menu for the GND symbol. Select the entry *GND Ground(3)* to obtain the EU-style symbol for ground (GND; chassis) rather than an earthing (PE) symbol.



Your circuit diagram should end up looking something like this.

24. Finally, save the circuit diagram. Select *Document/Save As* and in the window that opens enter any filename of your choosing under 'Title.' The other settings can be left as they are. It is also possible to publish your project so that other users can access it via the Internet by enabling 'Public.' Finally, click on 'Save.'



25. Using *Document/Export/...* it is possible to save your circuit diagram locally in a choice of data formats. If you want to use a monochrome graphics format, first select *Theme/Black on White*.

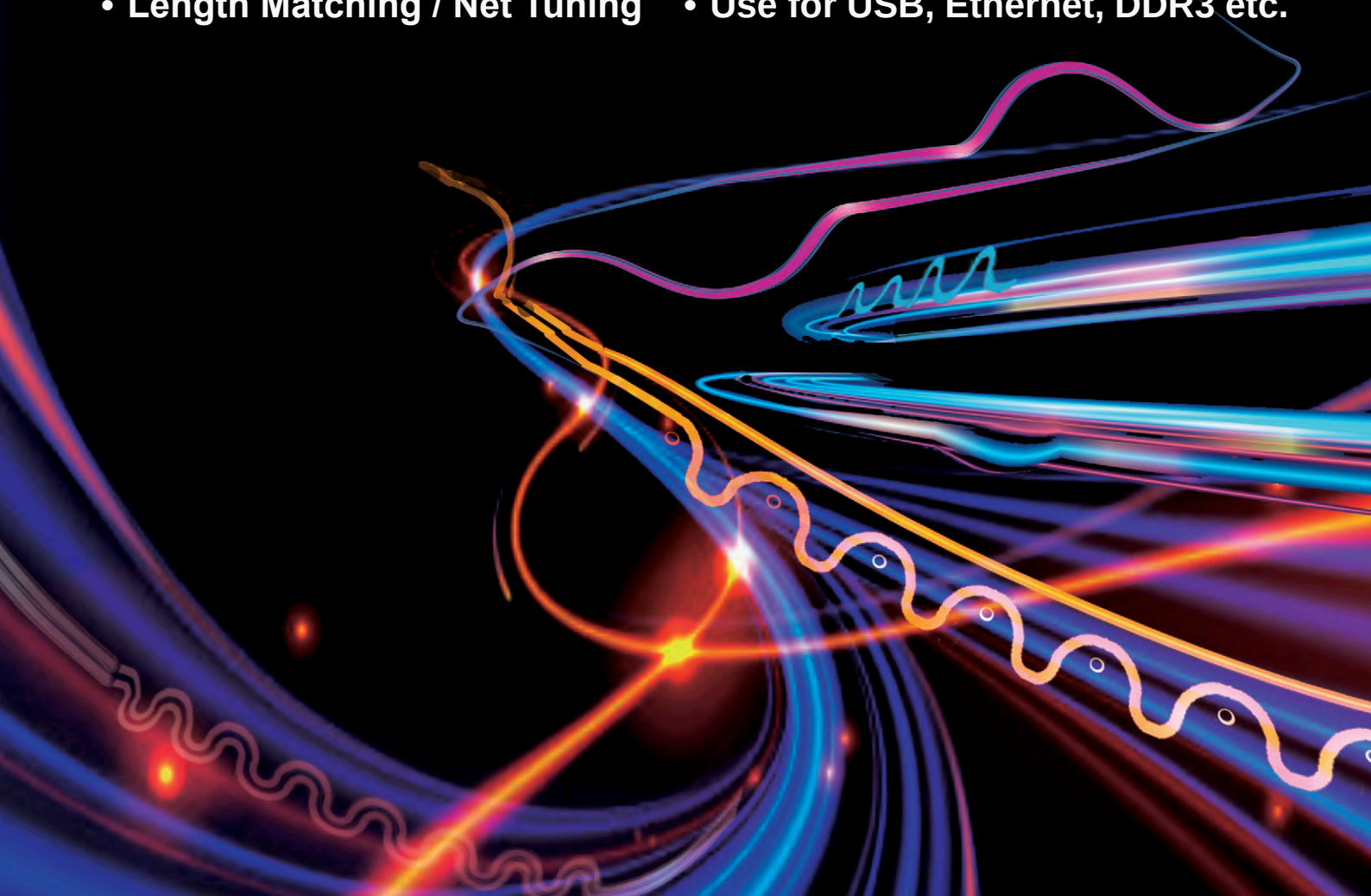
This article originally appeared in German in a special edition of Elektor offering an introduction to electronics based on the Arduino platform. This publication will be available in English from October 2020.

200225-02

PROTEUS DESIGN SUITE

High Speed Design Features

- Differential Pair Routing
- Length Matching / Net Tuning
- Automatic Phase Matching
- Use for USB, Ethernet, DDR3 etc.



Performance without the price premium.

Find out more and
configure your package on our
website or call the team on (+44)1756 753440

labcenter Electronics

www.labcenter.com

Join the Elektor Community

Take out a elektor GOLD membership!

- The Elektor web archive from 1974!
- 6x Elektor magazine (Print)
- 9x Digital (PDF) including Elektor Industry (EN) magazine
- A 10% discount in our web shop and exclusive offers
- Elektor's annual DVD-ROM
- An online Elektor LABs account, with access to more than 1000 Gerber files and a direct line of communication with our experts!
- Bring a project to publication or even sell it in our shop

Also available

The Digital elektor GREEN membership!

- Access to Elektor's web archive
- 10% discount in our web shop
- 6x Elektor magazine (PDF)
- Exclusive offers
- Access to more than 1000 Gerber files

www.elektor.com/member

elektor
design › share › sell