Edition 4/2019 | July & August 2019 | www.elektormagazine.com

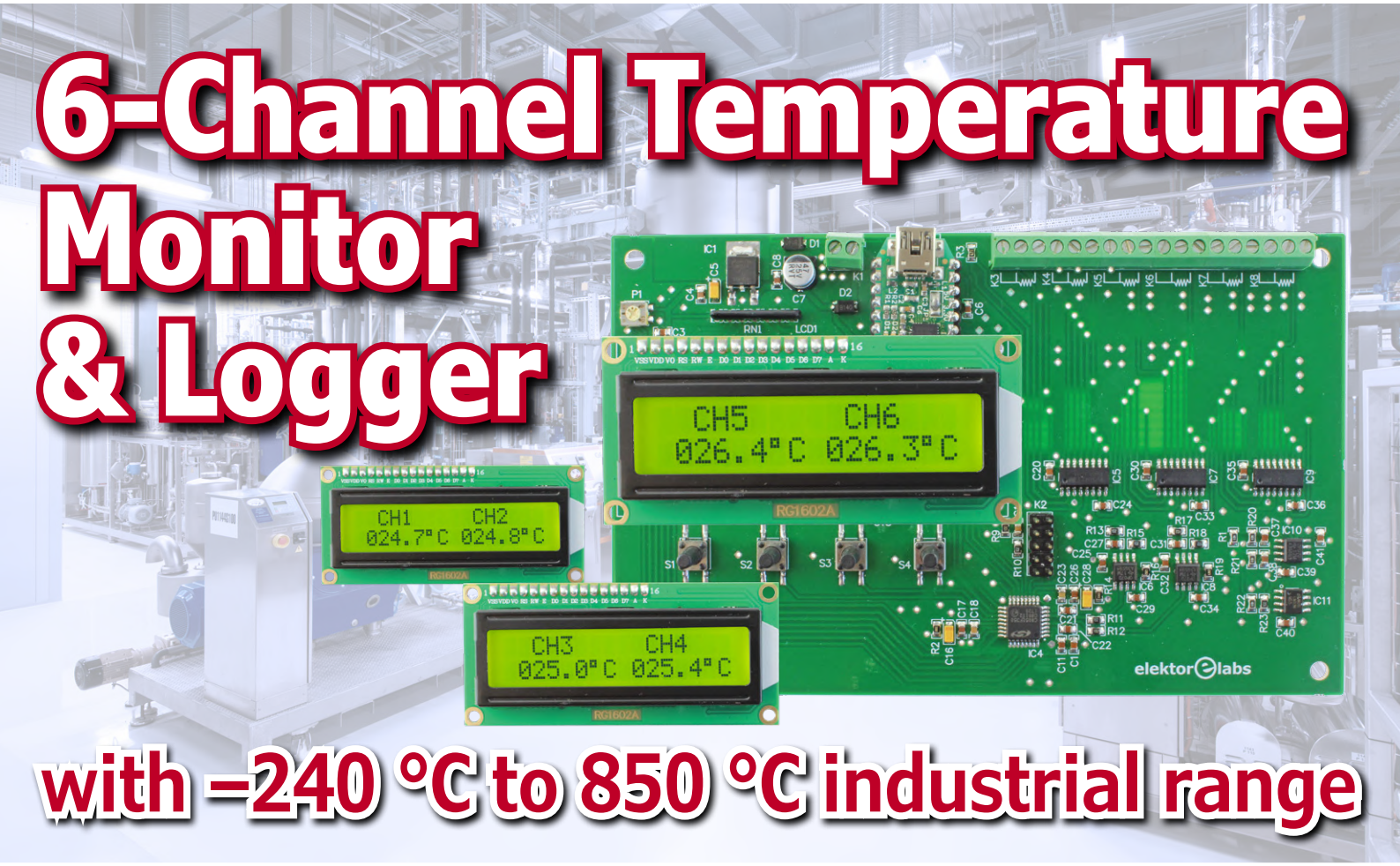
elektor

elektorlabs

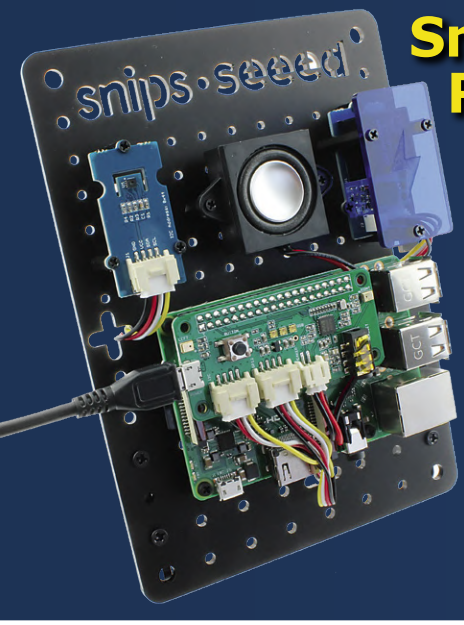
DESIGN BUILD SELL ELECTRONICS

magazine

6-Channel Temperature Monitor & Logger

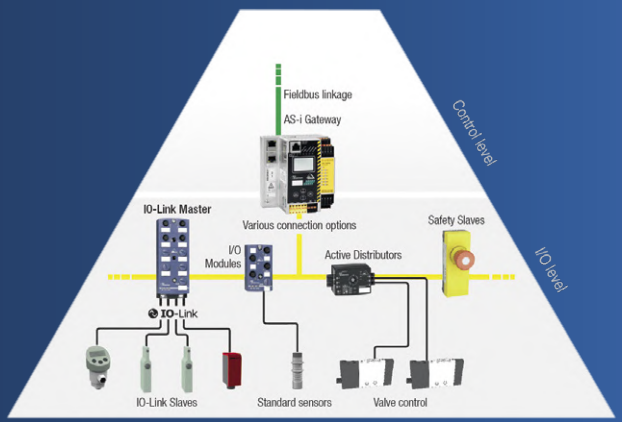


with –240 °C to 850 °C industrial range



Snips Speech Recognition

With an example for the Raspberry Pi



AS-Interface Essentials

Industrial automation bus system explained

6-Channel Temperature Monitor & Logger ⊳ AC-in, DC-out Baristors ⊳ An Elektor BoBs Catalog ⊳ AS-Interface Essentials ⊳ Bright Flashing LED ⊳ Create a 100% Private Vocal Assistant with Snips ⊳ Creating a New Component (Symbol) in KiCad ⊳ Differential Probe Power Supply ⊳ Display Serial Data on a Web Page ⊳ ESP32 as a Time Server ⊳ ESP32 Battery Monitor ⊳ Fair Algorithms ⊳ From Tubes to Silicon ⊳ Hexadoku ⊳ IOTA Crypto Currency (and an FPGA for the RPi) ⊳ MIDI I/O Breakout Board ⊳ and more...

DISCOVER OUR NEW PRODUCTS!

- **IP 67** Multimeter from **79,- €**
- **Graphic** Multimeter from **189,- €**
- **Bluetooth** Multimeter from **109,- €**

from 79,- €



All prices are RSP excl. VAT

- **Basic** Oscilloscope from **299,-**
- **Professional** Oscilloscope from **599,-**
- **Touchscreen** Oscilloscope from **699,-**

from 299,- €



Whether you are looking for professional digital multimeters with data logger and Bluetooth interface, 4-channel oscilloscopes with integrated generator, multimeter and CAN bus decoder or simply a low-cost basic oscilloscope – PeakTech® offers extensive new products at the best price!

www.peaktech.de

PeakTech®

Our value is measurable.

Elektorlabs Magazine
Edition 4/2019
Volume 45, No. 496
July & August 2019

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

www.elektor.com
www.elektormagazine.com

Elektorlabs 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
Fax: (+31) 46 4370161

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@eimworld.com

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

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

© Elektor International Media b.v. 2019
www.eimworld.com
Printed in the Netherlands



Wide, Wider, Wideband Scope for Electronics

For every edition of Elektor Labs magazine the editorial team makes a concerted attempt at balancing what effectively goes in print and online. In the process, various 'ranges' contend as we want to keep as many readers happy as possible without being sycophants to fans of just one field in electronics.
A weighting quadrant already arises by declaring the ranges *hobbyist – professional* as one "scale" (say *x*) and *100% hands-on – nice to know* as the other (*y*). But there are confounding '*z*' ranges as well, meaning more aspects to complicate our final compilation: old school *versus* programming only, microcontroller *versus* analogue, sponsored *versus* journalistic, technology *versus* ethics, and Homelab *versus* Elektor Labs, just to mention a few. And I mean *versus* in the sense of: contrasting with; not: opposing.
The pages you are about to read, skip, flip, scan or browse, are sure to reveal some of the extremes of the ranges I just mentioned and should help to define the middle ground. For example, for a truly ambitious project I would point at 'IOTA Crypto Currency (and an FPGA for the RPi)' on page 64, where cutting edge electronics — mostly microcontroller of course — is presented at a terrific pace. This project not just taps into the world of 'new trading' willy-nilly, but also explains how it all works without losing grip of the hardware. Compare that to the delightfully old-school 'Bright Flashing LED' on page 48 or the slightly more challenging 'AC-in, DC-out Baristors' on page 84 and you have a rough idea of the scope of 21st century electronics as well as that of the project proposals we receive through our Elektor Labs website and our editorial email accounts. Thanks all and keep 'm coming!
It's a vast scope and somehow terms like wideband, trigger slope, focus, intensity, overshoot, and sensitivity seem to apply to Elektor's dreaded selection criteria for article proposals. A tip: the authors in publication here got the trigger setting right. Which is another way of saying we work painstakingly, like a 'scope, to present crisp images on your screen.

Jan Buiting, Editor-in-Chief

The Circuit

Editor-in-Chief:	Jan Buiting
Translators:	David Ashton, Jan Buiting, Martin Cooke, Ken Cox, Arthur deBeun, Andrew Emmerson, Mark Owen, Julian Rivers
Membership Manager:	Denise Bodrone
International Editorial Staff:	Eric Bogers, Mariline Thiebaut-Brodier, Denis Meyer, Jens Nickel
Laboratory Staff:	Clemens Valens, Ton Giesberts, Luc Lemmens, Jan Visser
Graphic Design & Prepress:	Giel Dols, Jack Jamar
Publisher:	Don Akkermans

This Edition

Volume 46 – Edition 4/2019
No. 496 July & August 2019

6-Channel Temperature & Logger

with –240 °C to 850 °C industrial range





Regulars

63 Peculiar Parts, the series
TL431 Precision Programmable Reference

76 Questions & Answers
(Almost) everything you always wanted to know about...Analogue Design and Circuitry

80 Err-lectronics
Corrections, Feedback and Updates to published articles

82 Elektor Store Highlights
Equipping Your Maker Lab

90 HomeLab Helicopter
Wondrous Things Electronic Spotted From Above

104 Retronics
From Tubes to Silicon

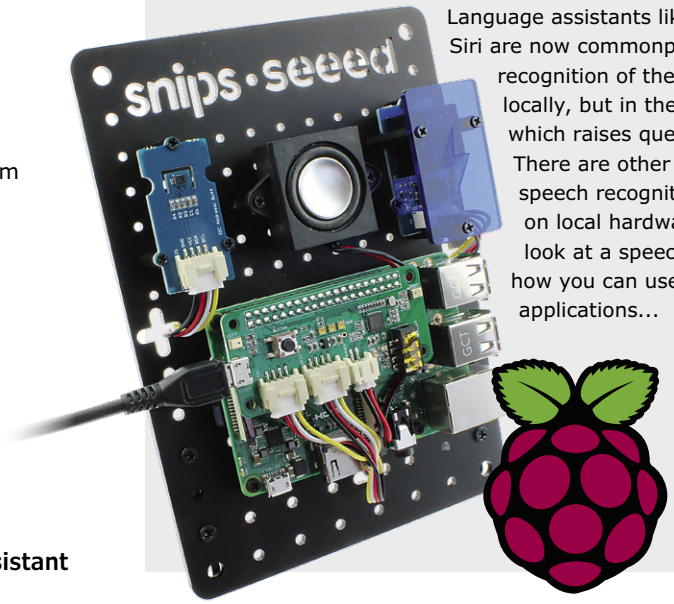
110 Elektor Ethics
Fair Algorithms

112 Elektor Store
what's available @ www.elektor.com

114 Hexadoku
The original Elektorized Sudoku

Snips Speech Recognition

with an example for the Raspberry Pi



Language assistants like Alexa, Google Home or Siri are now commonplace. However, the actual recognition of the spoken words is not done locally, but in the Cloud of the provider – which raises questions about data privacy. There are other solutions such as Snips – a speech recognition system that runs entirely on local hardware like the RPi. We took a look at a speech recognition kit and show how you can use it to develop your own applications...

22

Features

14 Create a 100% Private Vocal Assistant with Snips
Everyone's entry ticket to the world of Artificial Intelligence, with a Raspberry Pi

22 Snips Speech Recognition
with an example for the Raspberry Pi

34 Revolutionary Battery Pack Design from TUE/InMotion
(it's) Student Demonstration Time!

49 Heading Towards a Tsunami of Light
News from Chalmers University

52 Elektor Start-up Challenge - Paris 2019
An international launchpad for start-ups

64 IOTA Crypto Currency (and an FPGA for the RPi)
Part 2: PiDiver — FPGA board for fast calculations

72 AS-Interface Essentials
How the industrial automation bus system works

78 An Elektor BoBs Catalog
It's the little things...

104

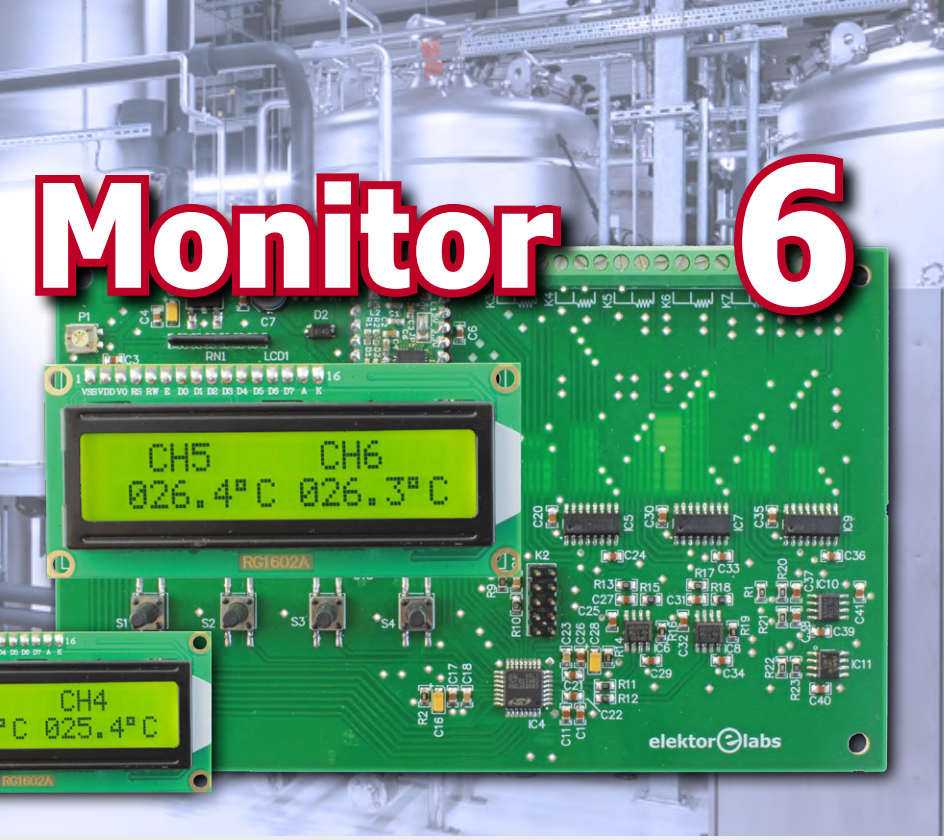
Retronics

From Tubes to Silicon



Projects

6 6-Channel Temperature Monitor & Logger
Measures from –240 °C to +850 °C



Monitor 6

elektor magazine

48 Bright Flashing LED
Traditional analogue...

50 Qt for the Raspberry Pi
A step by step guide to set up the GUI framework

54 New CPLD Board with MAX 10
More logic with the MAX 10 family

58 Waterflow Monitor with ESP32
With text/SMS alert of leaks

84 AC-in, DC-out Baristors
Buildings block for application in power supplies

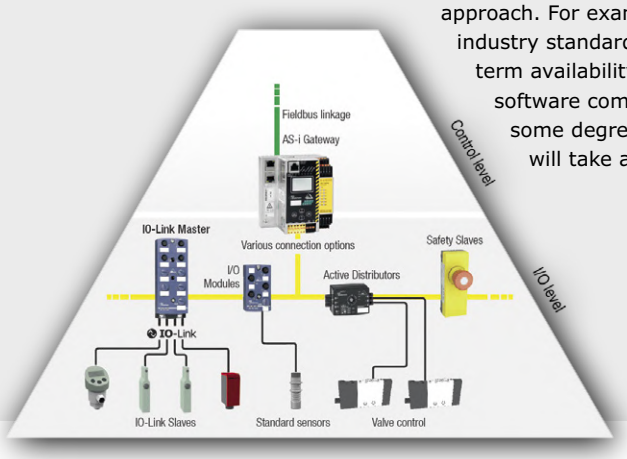
94 ESP32 as a Time Server
ESP32 plus RTC plus GPS plus display equals an NTP server on your own LAN

100 Creating a New Component (Symbol) in KiCad
and preparing to associate an existing footprint

AS-Interface Essentials

How the industrial automation bus system works

In the olden days everything was controlled using hand-made custom cabling and proprietary data formats. In the industrial world at least, however, this kind of thing is long gone: now entire systems are based instead around fully-featured bus architectures. As well as the obvious advantages of easier maintainability and expandability there are other reasons to prefer such an approach. For example, a widespread industry standard ensures the long-term availability of hardware and software components, at least to some degree. In this article we will take a look at AS-Interface.



72



26 MIDI I/O Breakout Board
Supports DIN and TRS connectors

29 The SCCC Project (3)
Homebrew soft-core processor and C compiler

38 ESP32 Battery Monitor
Measures three channels via Wi-Fi and Thingspeak

42 Differential Probe Power Supply
An ultra-simple power solution using USB

43 Display Serial Data on a Web Page
Using PHP or Python scripts

Next Editions

ElektorLabs Magazine Edition 5/2019

Horticulture Box • Build Your Own RFID reader Writer • 8 Channel Timer with OLED Display • Sound Blink DIY Portable Speaker • Flicker Free LED Dimmer • WSPR SDR • Q&A on Sensors • Littlevgl GUI library with ESP32 and a Raspberry Pi Touchscreen • USB-RS232 converter (FT231X BoB) • Remote Control for Hover-Catamaran • A Time-Corrected Woofer Concept • Universal battery Charger • Wireless In-Car Voltmeter.

ElektorLabs Magazine edition edition 5/2019 covering September & October 2019 is published on 22 August 2019. Delivery of printed copies to Elektor Gold Members is subject to transport.
Contents and article titles subject to change.

Elektor Industry Edition 3/2019

Elektor Industry issue 3/2019 is a special edition for the **Productronica 2019 trade show in Munich, Germany (November 12-15, 2019)**, and will focus on a range of innovative technologies, research, hardware and software, including IoT, 5G, sensors, CAD, SMT, PCB manufacturing, and more. Plus, you'll find fresh instalments of the magazine's regulars like Infographics and Industry Store.

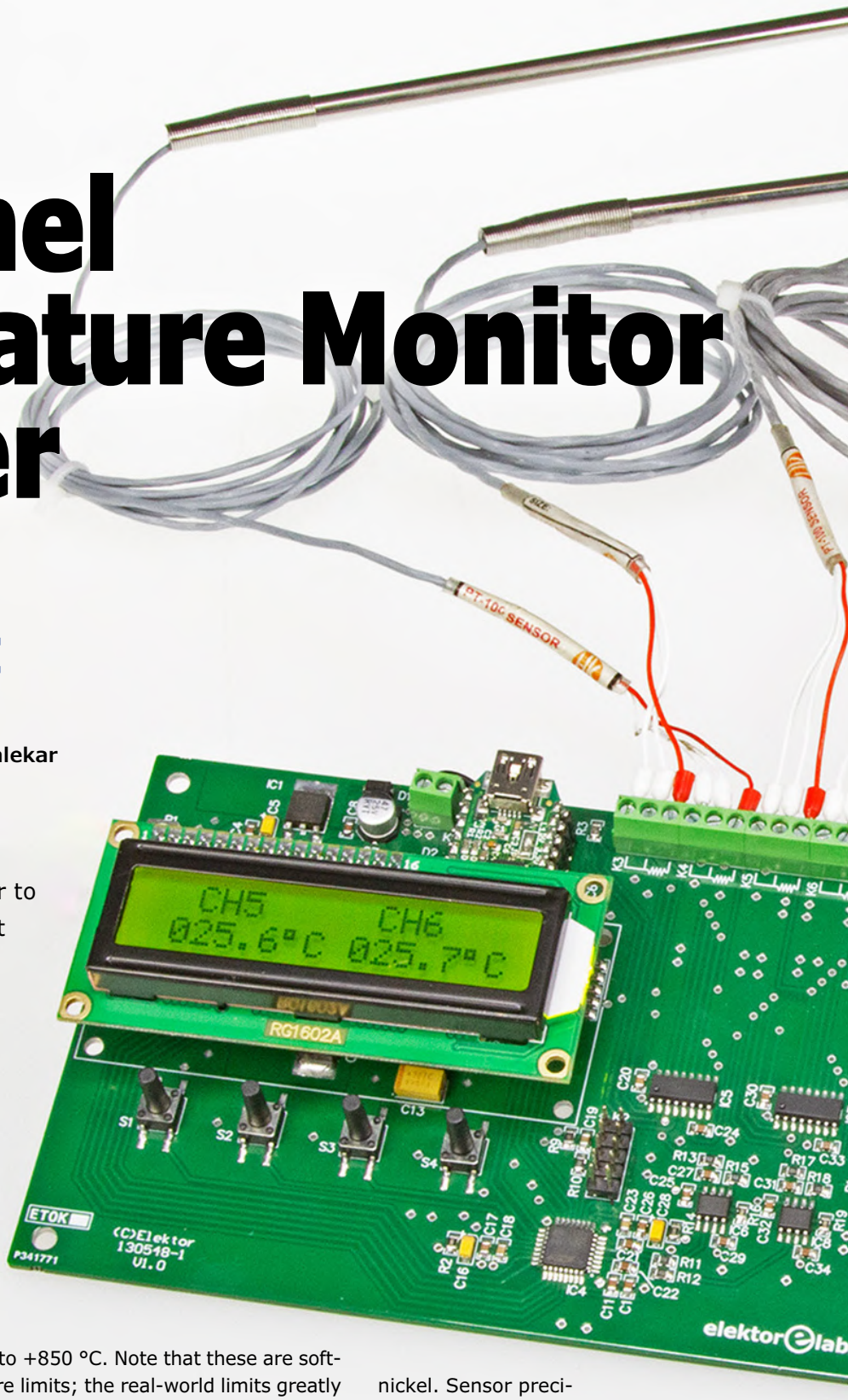
Elektor Industry issue 3/2019 will be published early November 2019 to Elektor Gold members in print and to Elektor Green members as a PDF download. The edition will be available for purchase at www.elektormagazine.com.

eLABS PROJECT

6-Channel Temperature Monitor & Logger

Measures from −240 °C to +850 °C

By **Clemens Valens** (Elektor Labs) and **Sunil Malekar** (India)



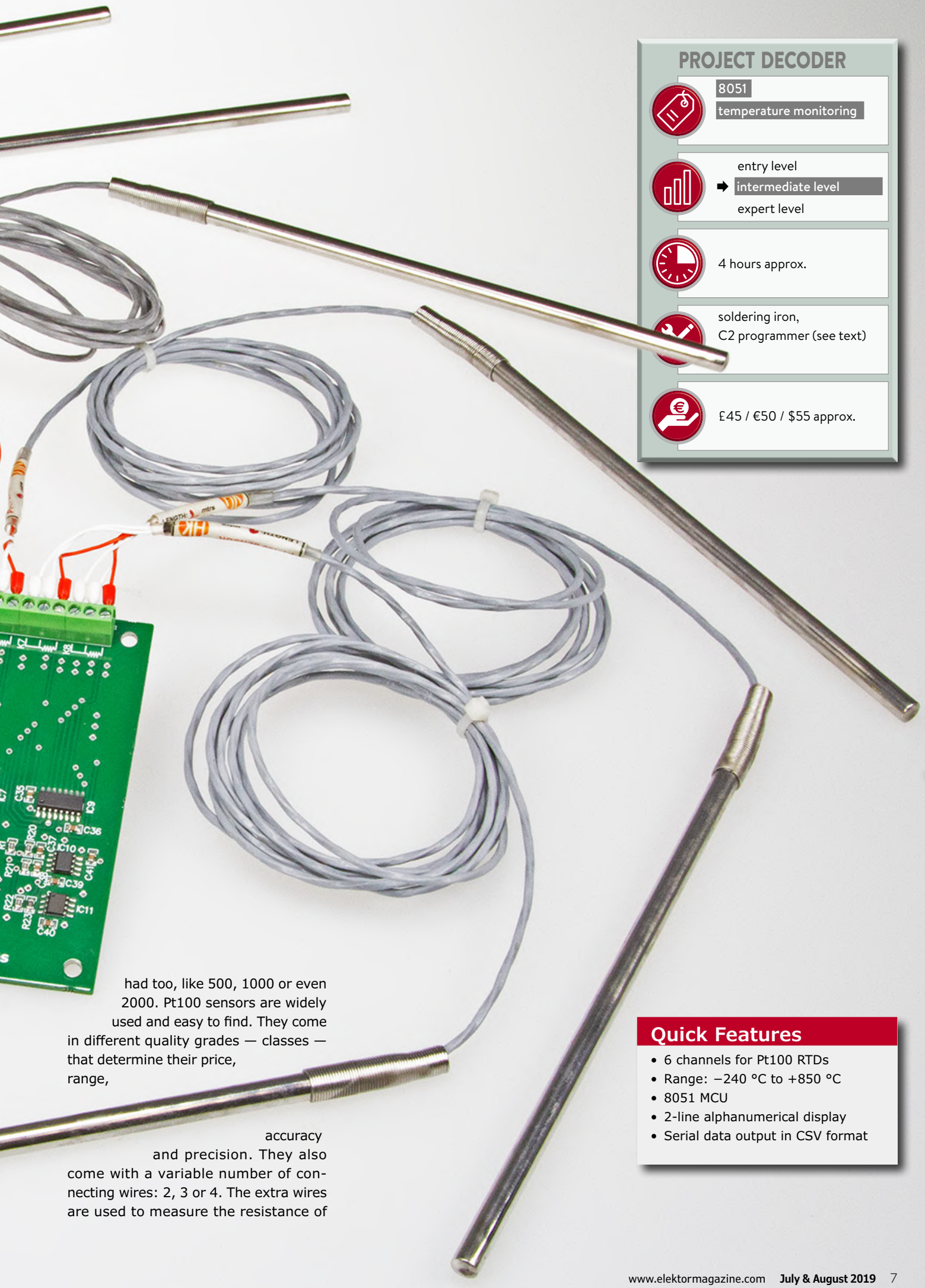
Temperature is an important parameter to control in many industrial processes. At home, too, the fridge, freezer, heater, kitchen oven, and air conditioner all measure and adjust the temperature to get their job done. Weather stations, cars, computers, tablets and smartphones keep an eye on temperature too. Come to think of it, temperature may well be the world's most surveyed physical quantity.

Even my low-cost Bluetooth speaker sounds an alarm when it gets too hot (as I discovered after placing it on a heater). Because humankind likes to measure temperature so much, we decided to design a device capable of measuring up to six of them. Usually, thermometers only measure one temperature — some can do two, often labelled 'inside' and 'outside' — but for process control more channels may be required. Range is also important, which is why the device presented here can measure temperatures across the massive range from −240 °C up to +850 °C. Note that these are software limits; the real-world limits greatly depend on the sensors being used.

Resistance thermometers

To measure temperature, we will use so-called RTDs in the shape of Pt100 probes. RTD is short for Resistance Temperature Detector, a device consisting of a length of metal wire with a known resistance/temperature relationship spiralled on a suitable supporting material. The metal typically used is platinum, copper or nickel. Sensor precision depends on the purity of the metal. Platinum has the most stable resistance/temperature relationship over the largest temperature range.

A Pt100 probe is an RTD made from platinum wire ('Pt' in the Periodic Table). The value '100' indicates that its resistance is 100 Ω at 0 °C. Other values can be



PROJECT DECODER

- 8051
- temperature monitoring
- entry level
- ➡ intermediate level
- expert level
- 4 hours approx.
- soldering iron, C2 programmer (see text)
- £45 / €50 / $55 approx.

had too, like 500, 1000 or even 2000. Pt100 sensors are widely used and easy to find. They come in different quality grades — classes — that determine their price, range, accuracy and precision. They also come with a variable number of connecting wires: 2, 3 or 4. The extra wires are used to measure the resistance of

Quick Features

- 6 channels for Pt100 RTDs
- Range: −240 °C to +850 °C
- 8051 MCU
- 2-line alphanumerical display
- Serial data output in CSV format

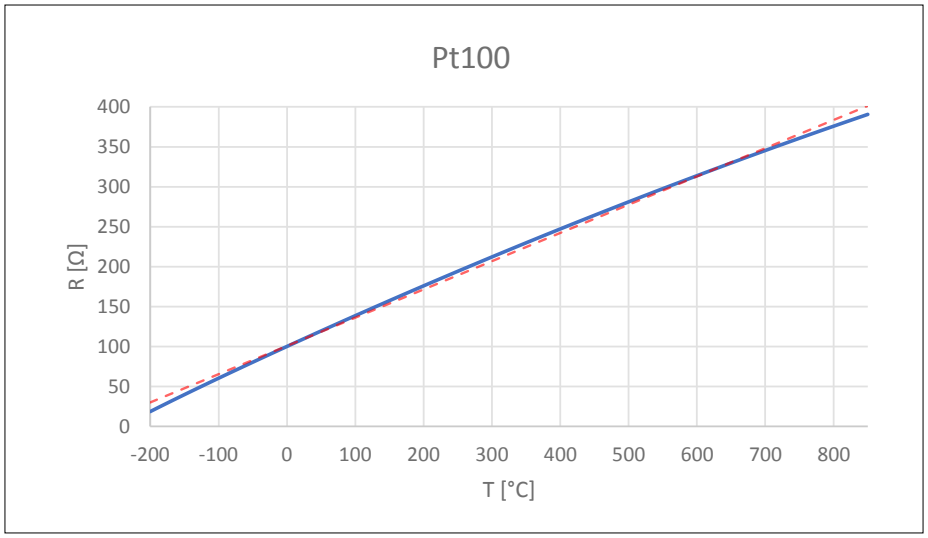


Figure 1: The relationship (in blue) between resistance and temperature of a platinum (Pt) wire is nonlinear. It may be approximated using a lookup table of short "straight" segments of the curve.

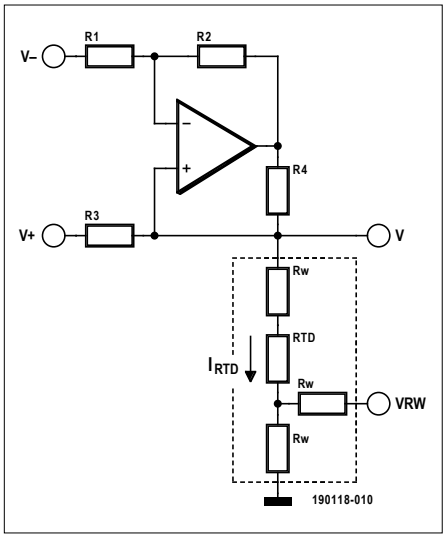


Figure 2: The Howland current source has a very high output impedance. It is used here to excite a 3-wire RTD.

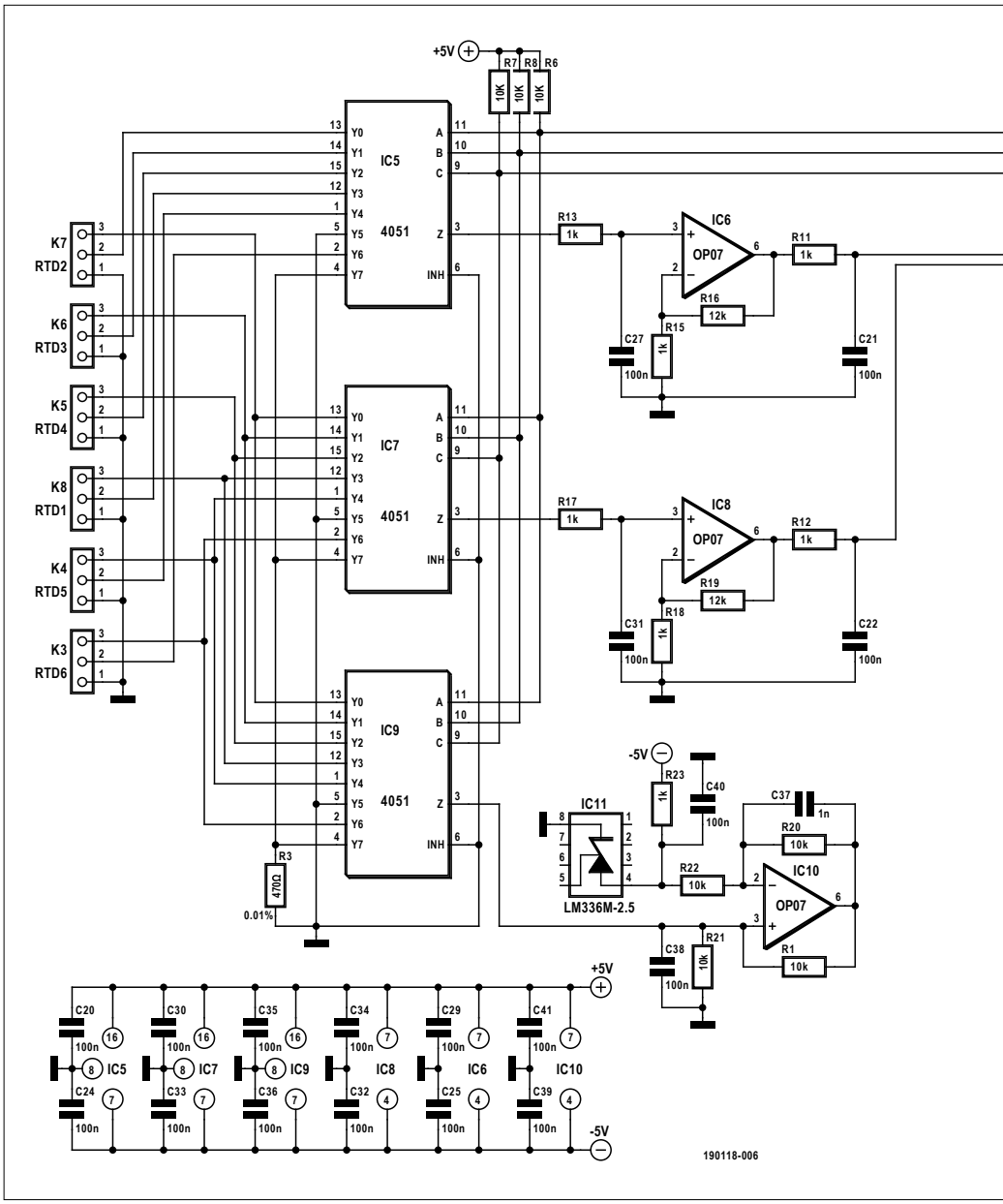
the RTD's connecting wires (more on that later). As these wires — that can be rather long — behave like inaccurate RTDs, it is important to compensate the sensor for them to ensure reliable and repeatable results. To avoid complicating our system too much, we opted for 3-wire RTDs.

An RTD is not a thermocouple

This is important to understand. A thermocouple is a junction of two different metals that produces a temperature-dependent voltage as a result of the thermoelectric (Seebeck) effect. Therefore, a thermocouple does not need a power source to operate whereas an RTD requires some form of excitation. Thermocouples tend to be less precise than RTDs, but they are very cheap.

Linearity

One last but important thing to mention about RTDs is their slightly non-linear resistance/temperature relationship (see **Figure 1**). This implies that some calculations must be done to obtain the correct temperature from the measured resistance. The Callendar–Van Dusen equation has long been used for platinum RTDs, but in 1990 the *Comité International des Poids et Mesures* replaced it by a 12th-order polynomial valid over the temperature range from -259.35 °C to 0 °C and a second, 9th-order polynomial for temperatures from 0 °C up to 961.78 °C. Because evaluating such polynomials calls for heaps of computing power (and debugging), most applications simply use lookup tables. We did so too.



Howland current source

As briefly mentioned above, an RTD must be excited in some way to make it usable. Either a constant voltage or a constant current source can be used for this. We preferred a constant current source since it poses fewer problems with long sensor wires. In this case the voltage across the sensor depends on the current only and, of course, the temperature, and not on the resistance of the connecting wires in addition.

Constant current source designs are plentiful but a popular one for use with RTDs is the Howland current source or current pump (**Figure 2**). This kind of source has the advantage of having a very high output impedance compared to other current source designs and it can sink as well as source current. However, to make it work as intended, resistors R1 to R4 surrounding the opamp should have a tolerance of 1% or better.

When the ratio R1/R2 is equal to the ratio R3/R4, then the output current is given by:

$$I_{RTD} = (V^{+} - V^{-}) / R3 \text{ [A]}$$

Compensating for connecting wires

Figure 2 also shows how the RTD is connected to the current source with its three wires. Assuming that the three wires are identical and that we can measure the voltages 'V' and 'V_{RW}' at a very high input impedance (meaning that there is no current flowing in the directions of V and VRW (properly: V_{RW}), then the voltage across the RTD is given as:

$$V_{RTD} = V - 2\,V_{RW} \text{ [V]}$$

This completes our theoretical introduction, and we can move on to studying the actual schematic.

The input multiplexer

Looking at the left of **Figure 3**, we find connectors K3—K8 for the six RTD channels. One wire of each RTD is connected to GND, the two others to a Type 4051 CMOS multiplexer IC. IC7 forwards the signal 'V' from Figure 2 to the rest of the system; IC5 does the same for signal 'VRW'. The third multiplexer, IC9, works the other way around and connects the output of the current source — constructed around IC10 — to the selected RTD.

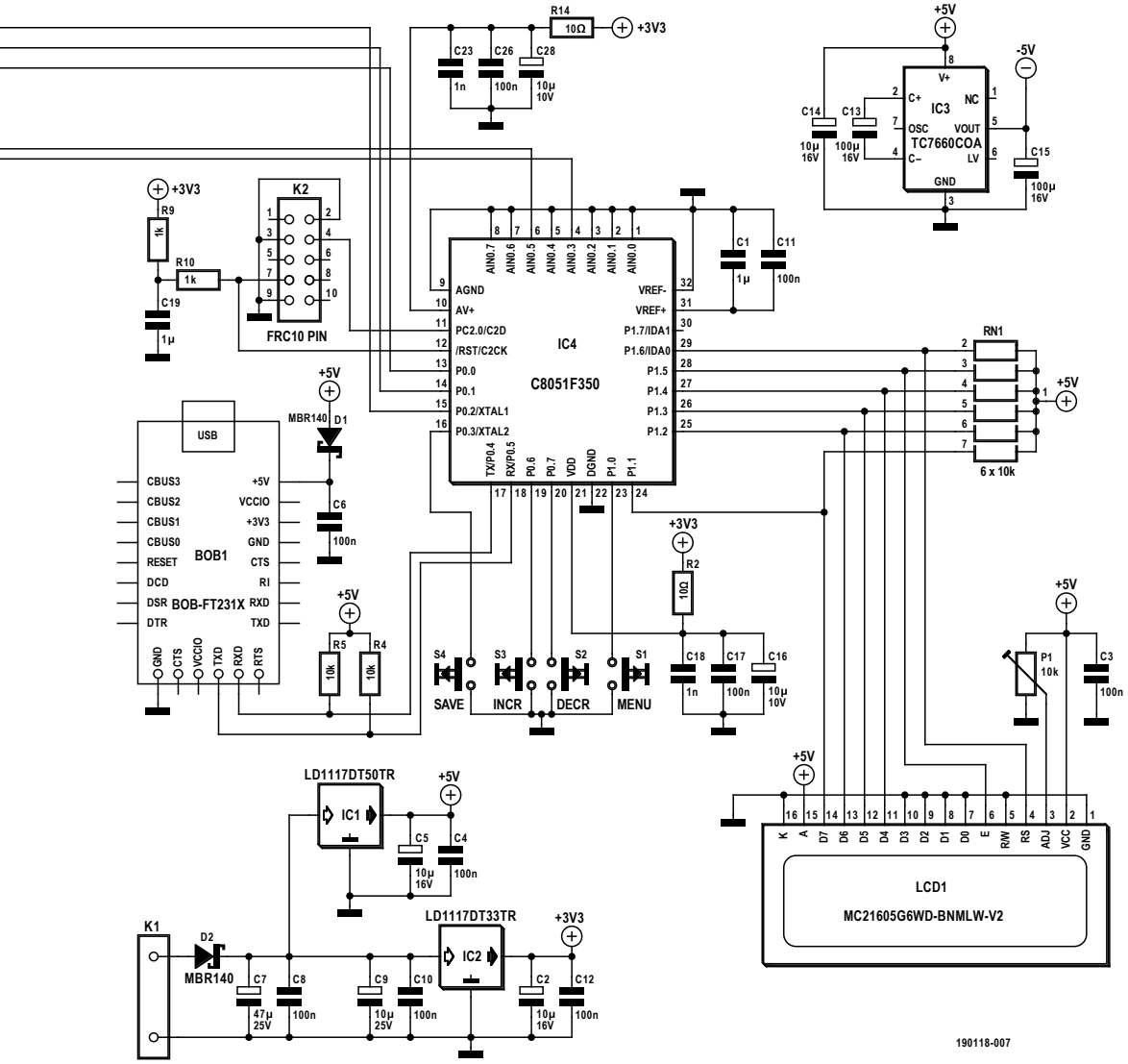


Figure 3: Schematic of the multi-node temperature logger. The analogue input stage is on the left, the digital part on the right.

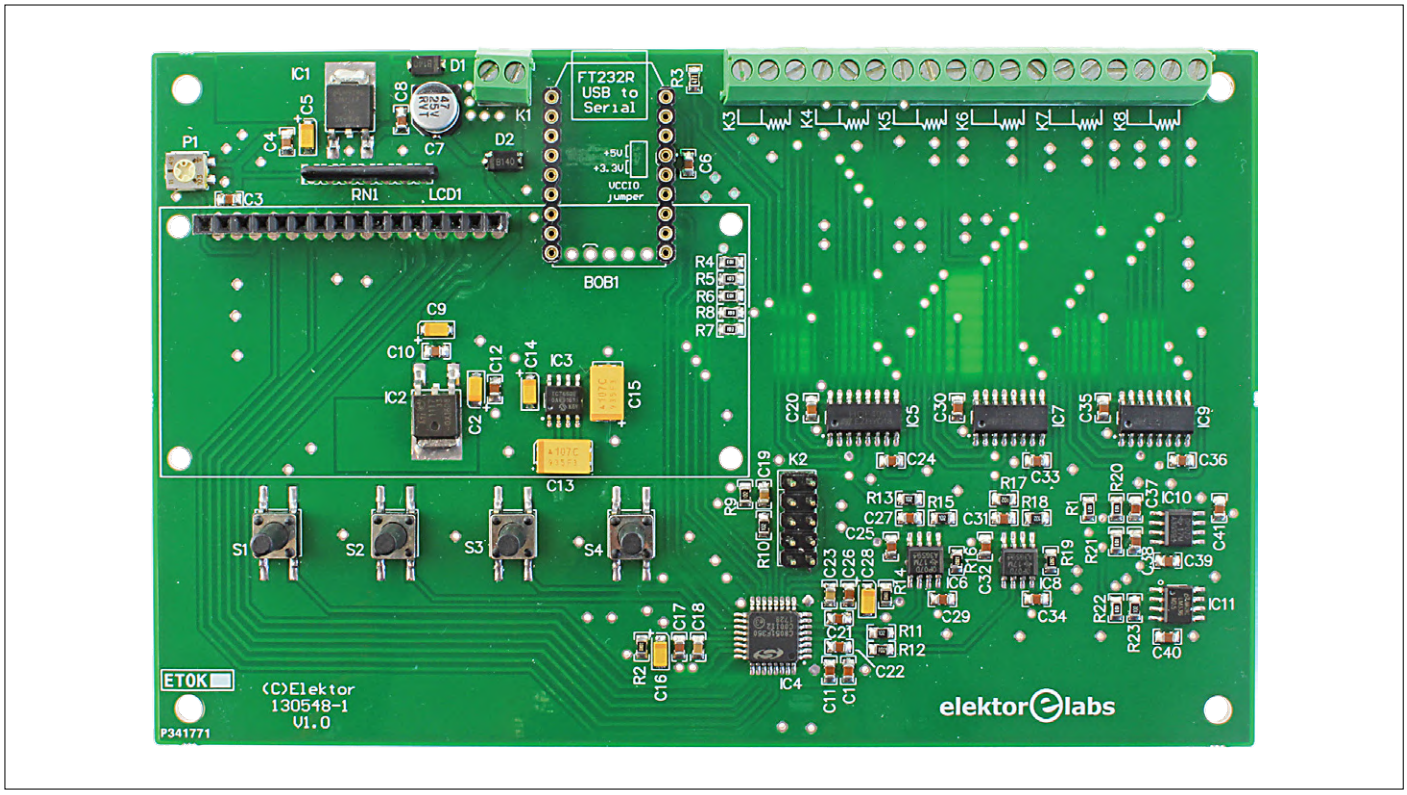


Figure 4: The assembled main board. Note: the board shown is version 1.0 that used the legacy FT232R BoB instead of the FT231X BoB.

Each multiplexer has eight channels, two of which are used for reference signals V_{min} (GND) and V_{max} (V_{R3}). Now you finally understand why there are six RTD channels and not seven or five. These reference signals follow the same paths as the RTD signals (R3, a 0.01% tolerance resistor, is even mounted close to the RTD connectors to mimic an RTD as close as possible). Because the values of these signals are known, this allows the software to calibrate the digitizer stage and remove errors introduced by the multiplexer and current source.

The outputs of the input multiplexer are individually amplified by a factor of 13 by IC6 and IC8. The signal 'V' from Figure 2 then arrives at the microcontroller's ADC input 3 (AIN0.3), while signal 'VRW' connects to AIN0.5.

Current source, part two

The Howland current source in the project comprises IC10, resistors R1, R20, R21 and R22. V^+ from our description above is connected to GND and therefore has a value of 0 V. V^- is equal to −2.5 V, produced by voltage reference IC11. R21 corresponds to R3 from the description above. The output current is:

(0 − −2.5) / 10,000 = +250 [µA].

Power supply

Theoretically, the supply voltage for the circuit connected to K1 can be up to 15 VDC, but this will put unnecessary strain on the low-drop voltage regulators. The minimum supply voltage is 7.5 VDC, the recommended value is 9 VDC. The total current consumption is around 70 mA including the 20 mA consumed by the backlight in LCD1. (Note that the latter has a built-in current limiting resistor.) IC1 lowers the supply voltage to 5 V for most of the circuit except microcontroller IC4; IC2 creates a 3.3-volt supply for this clever part.

The analogue input stage also requires a −5 V supply voltage, and this is taken care of by IC3.

The microcontroller

Over the past years, we have seen so many Arduino-based circuits flashing by that some have come to believe that there exists only one type of 8-bit microcontroller. That this is not true is proven by our circuit with its pimped-up, 8051-based brain! Now, before turning away in disgust, be aware that almost forty years after its introduction the 8051 is still a widely used 8-bit processor. Manufactured by Silicon Labs, the C8051F350 MCU used here runs internally at 49 MHz and executes most of its instructions in one cycle, making it a pretty fast device. The C8051F350 is part of a family of four that differ only by package type and ADC resolution. The 'F350 is the largest part with a 24-bit sigma-delta ADC, eight dedicated analogue inputs, and 17 digital I/O pins. It's an analogue-oriented controller with built-in digital filters and two 8-bit DACs. In this project we do not use all its capabilities, but you may want to study the MPU further for some future project.

Although running from 3.3 volts, the C8051F350's digital I/O pins are 5 V tolerant, which is why it interfaces nicely with the 5-volt-powered peripheral circuitry (note the pull-up resistors R4 to R8 and network RN1).

A quartz crystal or resonator is not called for as the MCU has a calibrated oscillator on chip.

Four pushbuttons (S1—S4) and an LCD provide the user interface. P1 sets the display contrast level; if you forget to adjust it when you power the board for the first time, the display may not show anything.

For communication with a computer, we added the option to plug a USB-to-

TTL-serial converter onto the board. We used our own breakout board, but it is also possible to connect a functionally equivalent interface. Please note that the photographs in this article show a prototype which used Elektor's (golden oldie) FT232R BoB, no. 110553. Since then it got replaced by our new FT231X BoB no. 180537 which is compatible with the *de facto* standard of 'FTDI' USB-to-serial cables.

Software dealings

At the time of writing the kind people of Silicon Labs still handed out free licenses for Keil µVision5 so people who develop software for the C8051 devices in C can do this in a comfortable way. When you request such a license it seems to be valid for just one month, but once installed in the IDE it becomes a permanent license. Of course, we have explored this avenue, and everything went fine — yet you never know when they decide to stop doing this and so we ported our software to SDCC, the open source compiler for 8051 and PIC devices.

SDCC does not work with projects and does not make use of any other GCC-based tool. Our project therefore is just a bunch of files in a folder together with a batch file to compile them.

As with every C program, the visible part starts in the function `main`. The first thing to do here is to switch off the watchdog timer; you can always switch it back on later when you are ready for it. (Note that the Keil compiler does this in the runtime (CRT) startup file so you don't notice it. We found that out the hard way.)

After initializing the MCU's peripherals — 1 ms system tick timer, UART running at 9600 baud, etc. — the program performs a calibration of the ADC. This is a pretty neat feature of the MCU as it allows us to calibrate away many of the imperfections of our analogue front end. After calibration, converting V_{max} — the voltage over 0.01% resistor R3 — will read as `0xffffff` (24-bit maximum value) while V_{min} (GND) will produce a reading of `0`. The ADC applies the corrections in hardware; nothing to do for us in software.

A word on sampling

The rest of the time `main` will spin in an endless loop, scanning for key presses and updating the display when needed. Every second a sample is taken from a channel, converted to a value in degrees Celsius or Fahrenheit and printed to the screen. A second later the next channel in line is updated. In the background the ADC runs at a faster rate of nearly 18.7 Hz. This higher speed is necessary for two reasons. First, to measure one RTD two voltages ('V' and 'VRW') have to be converted as the RTD value must be compensated for the voltage over the connecting wires. Second, the ADC has a built-in low-pass filter that needs three samples to calculate an output value. Therefore, at least six samples are required for one RTD measurement, which corresponds to 320 ms. By reducing the display time to 1 Hz, the system has enough slack to sample and filter comfortably.

Lookup table

Samples must be converted to temperature values and this is done with the help of a lookup table. Pt100 conversion tables can be found on the Internet and we compiled one from −240 °C to +850 °C with a 10-degree step size. Below −240 °C things become highly inaccurate and so we limited the display to −260 °C (which actually means Underflow). For similar reasons the upper limit was fixed at +850 °C.

Of course, most ADC readings will be in between two table values, hence linear interpolation is used to estimate the temperature. If more precise conversions are desired, it is possible to make the lookup table's step size smaller, as long as it fits in memory (of which there is not a lot). If the table becomes really large and unwieldy, you might want to implement a more efficient search algorithm than the linear search we used or use a polynomial approximation.

Construction

Even though the project uses mostly SMT parts, assembling the PCB should not be too difficult. Start by mounting 2-terminal parts like resistors and ceramic capacitors and then continue with the taller parts. Respect and verify the polarity of polarized parts like diodes and electrolytic capacitors.

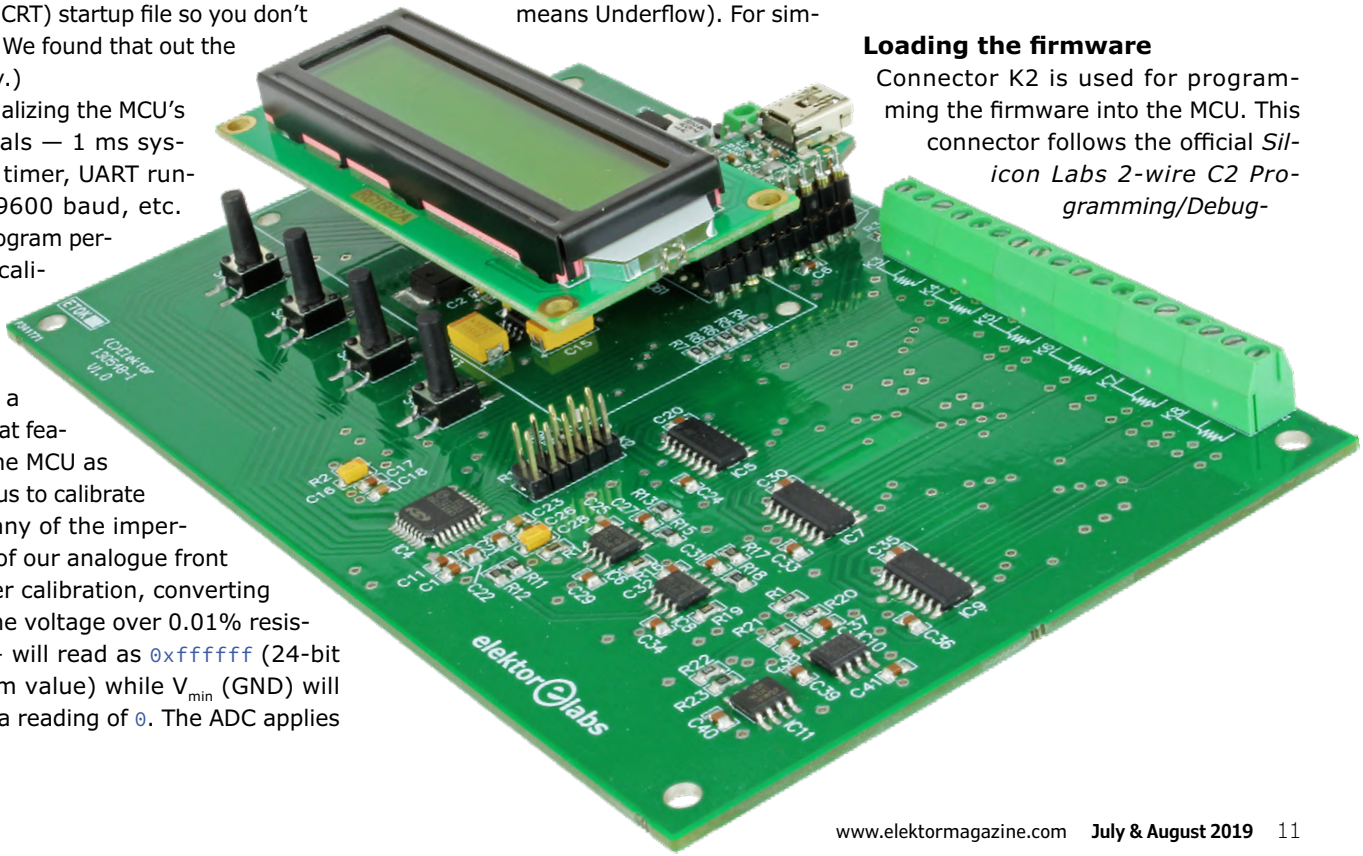
It is always a good idea to build and test the power supply (D2, IC1 and IC2 and the capacitors around them) before mounting the other parts.

Make sure to use resistors with 1% tolerance in the input stage, except R3 which must have a tolerance of 0.01%. To create a real good current source, you might want to use 0.1% tolerance resistors for R1, R20, R21 & R22.

Depending on if and how you mount the USB-serial interface, you may want to use extra-tall connectors for the display as it is mounted over the serial BoB.

Loading the firmware

Connector K2 is used for programming the firmware into the MCU. This connector follows the official *Silicon Labs 2-wire C2 Programming/Debug-*



ging Interface Standard so connecting a C2-compatible programmer should be easy. Less easy, on the other hand, is getting your hands on such a programmer, which is why we rolled our own. The free download on the support page for this article [3] includes an Arduino sketch that transforms any Arduino-Uno-compatible board into a C2 programmer. You can also find it on GitHub [2]. The MCU's 5 V-tolerant pins allow us to do this with impunity.

Connect Arduino pin A0 to K2 pin 7 ('C2CK'), Arduino pin A1 to K2 pin 4 ('C2D'), and Arduino GND to K2 pin 2. Connect the Arduino to the computer before powering the board and all should be fine.

A Python 3 script, also included in the download, permits uploading Intel HEX files to the home-brew programmer. This script requires pySerial.

Using the system

There are two firmwares for this project, one with debugging options ('d' suffix) and one with fewer options. The main UI functions are the same for both.

- 'Menu' (S1): press this button to enter the settings "page". On this page you can toggle the temperature unit between Celsius and Fahrenheit by pressing S1 again. With the keys 'Decr' (S2) and 'Incr' (S3) you can adjust the display refresh value from one second to one minute. When done, you can either press 'Save' (S4) to store the new settings in non-volatile memory or let the page time out. When the page times out the new settings will be used until the next change or a power cycle.
- 'Save' (S4): press to recalibrate the system.

COMPONENT LIST

Resistors (0805, 1%)

R4,R5,R6,R7,R8 = 10kΩ
R1,R20,R21,R22 = 10kΩ 0.1%
R2,R14 = 10Ω
R3 = 470Ω 0.01%
R9,R10,R11,R12,R13,R15,R17,R18,R23 = 1kΩ
R16,R19 = 12kΩ
P1 = trimmer 10kΩ
RN1 = resistor network, 6×10kΩ

Capacitors (0805)

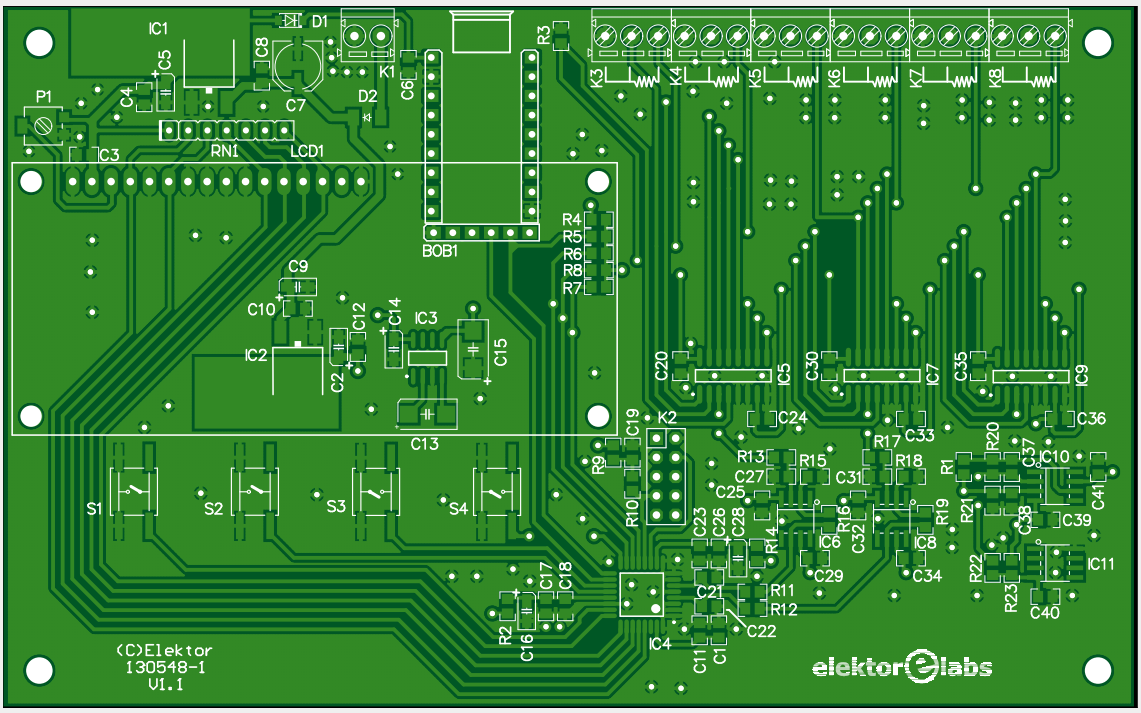
C1,C19 = 1µF
C2,C5,C9,C14,C16,C28 = 10µF, 16V, Case-A
C3,C4,C6,C8,C10,C11,C12,C17,C20,C21,C22,C24,C25,C26,C27,C29,C30,C31,C32,C33,C34,C35,C36,C38,C39,C40,C41 = 0.1µF
C7 = 47µF, 25V, 6.3mm diam.
C13,C15 = 100µF, 16V, Case-C
C18,C23,C37 = 1nF

Semiconductors

D1,D2 = MBR140SFT1G
IC1 = LD1117DT50TR
IC2 = LD1117DT33TR
IC3 = TC7660EOA
IC4 = C8051F350-GQ, LQFP32
IC5,IC7,IC9 = 4051, SOIC16
IC6,IC8,IC10 = OP07CD
IC11 = LM336M-2.5

Miscellaneous

K1 = 2-way wire-to-board terminal block, 3.5mm pitch
K2 = 10-pin, 2-row pinheader, 2.54mm pitch
K3-K8 = 3-way wire-to-board terminal block, 3.5mm pitch
LCD1 = Alphanumeric LCD, 2 lines, 16 characters, 5V with backlight
S1-S4 = Tactile switch, 6×6.2mm, SMD
BOB1 = USB/Serial Bridge/BOB with suitable connectors
16-way pin socket + pinheader, 2.54mm pitch for LCD1
Elektor PCB # 130548-1



The debug firmware adds to this:

- 'Decr' (S2): pressing this key during startup and keeping it pressed until the splash screens have passed will skip ADC calibration at startup. Use 'Save' (S4) to calibrate when you want to.
- 'Decr' (S2) and 'Incr' (S3): pressing these keys at the same time will open the debug page where you can see raw 16-bit ADC values for both the signals 'V' (RTD) and 'VRW' (Wire). Pressing 'Incr' will select the next channel. Press 'Save' (S4) to return to normal mode.

Data logging

Plug BOB1 on the board (or use another USB-to-serial converter, the connections are compatible with an 'FTDI' cable) and connect it to a free USB port on a PC. The latter should detect it as a serial port. Launch a terminal program and configure it for 9600 baud, eight data bits, no parity and one stop bit (9600n81). As soon as the board is powered, it will start sending comma-separated values (CSV) in human-readable ASCII format. Most serial port terminal programs will allow you to record the data to a file so it can be processed further in for instance a spreadsheet. ⏮

(190118-01)

Web Links

[1] Project at Elektor Labs: www.elektormagazine.com/labs/6-channel-temperature-monitor-logger

[2] C2 Programmer at GitHub: https://github.com/ElektorLabs/ElektorLabs_Tools

[3] Web page for this project: www.elektormagazine.com/190118-01

@ WWW.ELEKTOR.COM

→ 6-Channel Temperature Monitor & Logger, partly assembled module
www.elektor.com/130548-91

→ 6-Channel Temperature Monitor & Logger, bare PCB
www.elektor.com/130548-1

→ LCD 2x16 characters w. blue backlight
www.elektor.com/120061-77

→ FT231X USB/Serial Bridge/BoB
www.elektor.com/180537-91

Advertisement



ARTIFICIAL INTELLIGENCE

Create a 100% Private Vocal Assistant with Snips



Everyone's entry ticket to the world of Artificial Intelligence, with a Raspberry Pi

By **Mennad Yami** and **Robin Guignard-Perret** (France)

Snips is a platform for connected "things". It allows developers and manufacturers of "things" to add a voice assistant to their products Your Raspberry PI may not yet obey your finger or your eye, but thanks to Snips it is at your command. This article explains how to add "ears" to a Raspberry Pi.

Formed in 2013, Snips wants to install an artificial intelligence assistant in every object , making the technology so intuitive that it will fade into the background.

The Snips technology is unique in that it executes locally on the actual object that the user is talking to, which implies that no data is ever sent to the cloud. It's a concept that guarantees confidentiality and continued functioning, even in the case of an Internet fault, which makes Snips one of the foremost voice technologies to follow the general standards for data protection. The Snips platform is free to anyone who wants to evaluate our technology for non-commercial use. We have created a community of 25,000 people - the biggest community of voice application developerartifs outside Amazon and Google. Snips works on Linux (Raspberry Pi), Android, iOS, MacOS and Debian. Anyone can discover online the collection of tools and documentation to create their own voice applications - see [1] Snips is available in French, English, German, Japanese, Spanish and Italian.

Using our models in embedded systems

The principle of privacy by design (simply, taking privacy into consideration right from the start) puts confidentiality at the heart of the system design process. This confidentiality is even more crucial as voice assistants are deployed in various scenarios in the private space of users. This choice guarantees users against any abuse, present or future, of their data. At Snips, "privacy by design" translates into the absence of any communication of user data to the cloud.
During the design of the Snips voice platform, priority was given to portability and good resource management. The system works on as light a platform as the Raspberry Pi, a common IoT platform board popular among developers. Other Linux boards are also supported, and the Snips SDK for Android works with the Android 5 and ARM environment, while the iOS SDK targets iOS version 11 or higher. For reasons of efficiency and portability, the algorithms are re-implemented whenever necessary in Rust (a compiled programming language from Mozilla Research).

Vocal assistants

Modern voice assistants can be broken down into several building blocks: the detection of a key word or *wakeword*, recognition of the spoken word (Automatic Speech Recognition, ASR), the understanding natural language (Natural Language Understanding, NLU) and an action code which will execute as soon as a command is detected.
The *wakeword* is the word that lets you indicate to your assistant the start of a command. A Snips assistant will start listening as soon as it hears "Hey, Snips!". It then begins to transcribe the words of the user, this is the function of the second building block: the ASR.
The **ASR** block tries to transform the sound signal captured by the micro into text intelligible to humans. If the user says "Hello world" the sound signal is first processed by a deep learning model, an acoustic model, which converts it into phonemes, the sound representation units used by linguists. "Hello world" is first perceived in the form "hɛˈləʊ wɜːld". Following this, a specific model for the language of the assistant is applied to transpose these phonetics into the vocabulary that we use, that is the model of the language. If the assistant is configured for English, "Hello world" will be correctly returned, then sent on to the next block, the NLU.
The NLU's purpose is to extract the meaning of a text, A voice assistant is designed to respond to a limited range of commands. The task of the NLU is to label a sentence with an intention. These intentions often come with parameters which should be isolated in order to correctly process the command. for example, if the user says "Hey, Snips! Switch on the light in Paul's room in red", when the NLU receives the transcription of this command, it distinguishes the intention "Switch on the light' with the parameters of place "Paul's room" and the colour "red". It is this information which will be redirected towards the action code.
The final building block is thus the **action code**, which waits, via an infinite loop, for an intention to be detected so that it can execute it, taking the received parameters into account. This code is specific to each application and each device, and can process any scriptable action. It can call on remote APIs, send commands to connected objects, speak via a voice assistant, or all these at once. The *wakeword*, the ASR and the NLU are the cognitive senses of your assistant, the rest is thus carried out by the action code (see the overview in **Figure 1**)
The existing voice assistants on the market (Google Home, Alexa, Siri) have their wakeword detector, their ASR and their NLU in the cloud. Their models, being very large, need far greater processing power than is available in embedded technology. This is not the case with Snips. The models are created on a case-by-case basis, for each assistant and each user, and are specific to the tasks for which they are designed. This is how, thanks to Snips, ultra light voice assistants can be created, capable of running on a Raspberry Pi Zero or 3, or more 100% offline assistants. All this is accomplished without any sacrifice in respect of voice recognition performance.

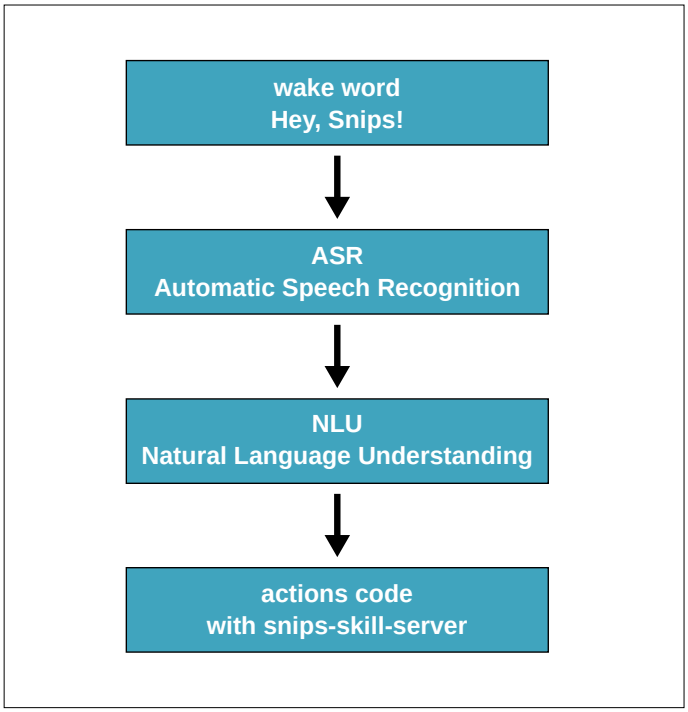


Figure 1: Overview of the building blocks needed for voice recognition.

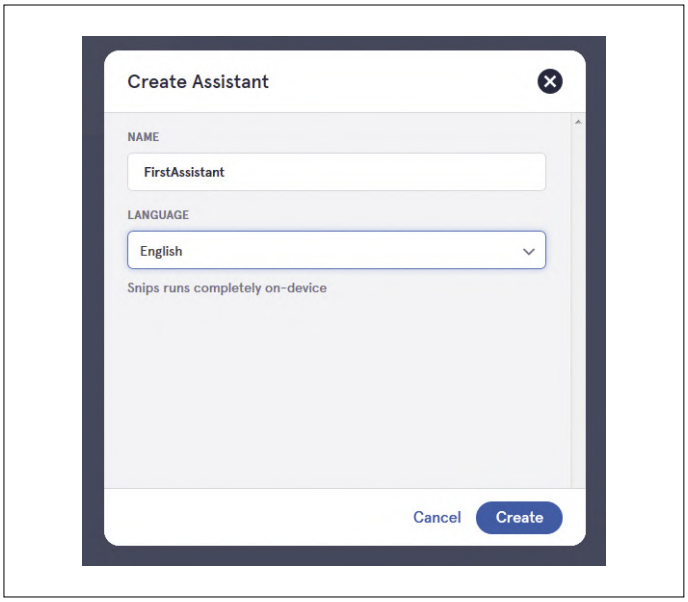


Figure 2: Creation of an assistant on the Snips online console.

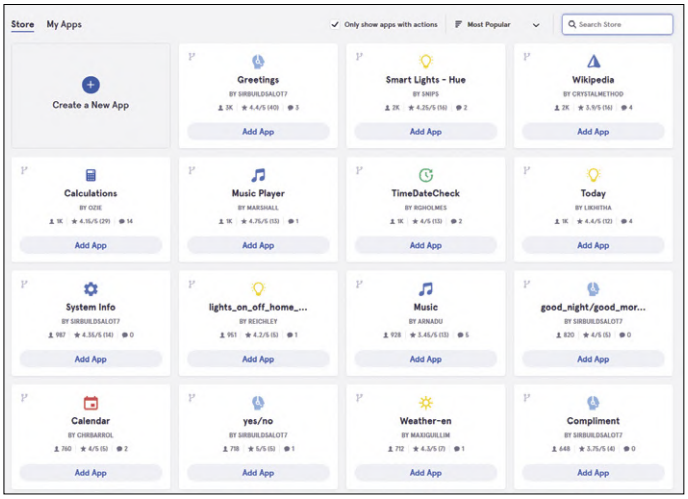


Figure 3: Examples of *apps* available on the console.

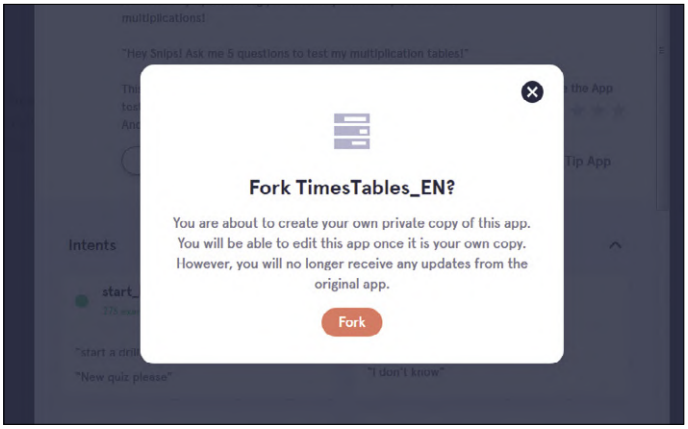


Figure 4: The *fork* function allows the creation of a copy of an existing *App* to modify it (this message appears automatically on clicking *Edit*).

The SNIPS platform

The Snips platform is an environment which comprises all the necessary parts of a working voice assistant. It is the installation of this which allows anyone to take advantage of his own voice assistant, respecting privacy, on Linux, MacOS, iOS and Android. Alongside the *Wakeword* detector, the ASR and the NLU, we find the Dialog Manager handles the coordination and succession of tasks presented to it. The **TTS**, or Text To Speech, handles voice synthesis for eventual responses. It receives text and reads it aloud, adapting its accent to the language of the assistant. Finally, the **snips-skill-server** is the component which handles the execution of the action code.

The internal communication of the platform uses the Hermes protocol, created by Snips around the MQTT protocol, light and quick. Thanks to Hermes, it is easy to customise the working of the platform to, for example, communicate with peripherals of your main equipment, make it say a sentence through the TTS, or command one of the platform components separately. Once the Snips platform is installed on a device, is ready for an assistant to be loaded. It is on the Snips web console [1] where this assistant is created (**Figure 2**). From the console the user can define the language of the voice assistant, its range of understanding and the functionalities (or apps).
For example, the platform, installed on your Raspberry Pi, can accommodate a French assistant with an App for control of lights, as well as a weather *App*.

Create and install the assistant

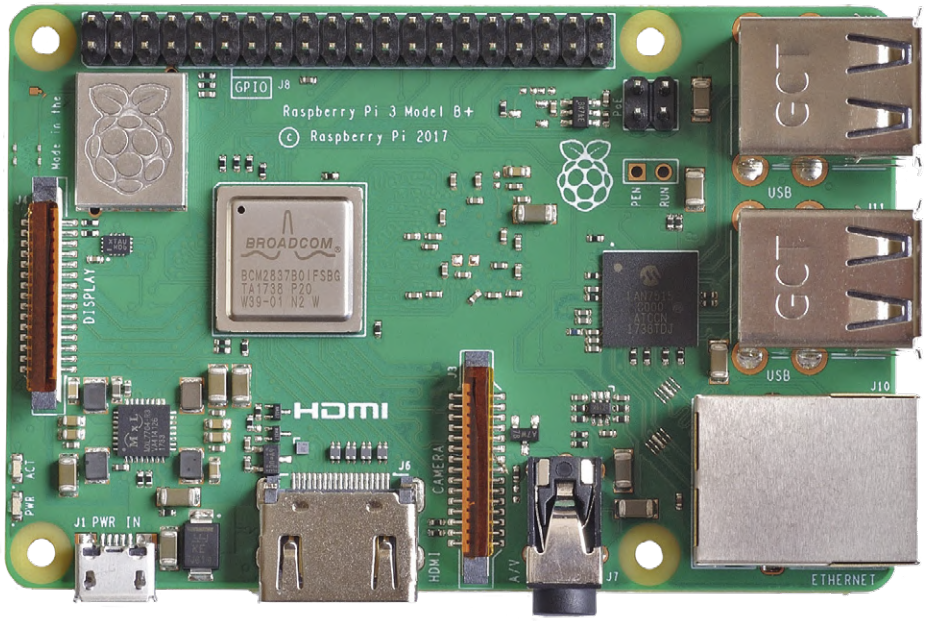
Let's pass now to the creation of your personal assistant, to deploy on an RPi. Your RPi must first of all accommodate the Snips platform. To ease the installation process and the management of assistants, we have created SAM (*Snips Assistant Manager*), and easy-to-use deployment tool run from the command line. Sam is a program for your computer, which connects to your RPi via SSH. The two devices must of course be on the same network, so think about connecting your RPi to a wireless network or direct to your computer by Ethernet. In addition, your main computer must support *Node.js* (JavaScript execution environment) and *npm* (packet controller of Node.js) to be able to run SAM. The command `sudo npm install -g snips-sam` is all that's needed to install the tool. Then the command `sam devices` which lists all the RPi present on your network will allow you to detect yours. Once you have obtained the IP address or the *hostname* of your device. connect it to SAM with the command `sam connect <adresseIP/hostname>`. Finally, `sam init` starts the installation of the Snips platform. To connect your microphone and your audio output, use `sam setup audio`.

Snips Console

Then go to [1] to create an account, then a new assistant. This assistant may have one or many function(s), you can choose from the *Apps* available in the App Store (Weather, recipe book, control of Philips Hue lamps... See **Figure 3**) or construct your own. You can also personalise an App from the App Store to produce a unique version thanks to the *fork* function (**Figure 4**).

An *App* captures intentions, or *intents*, thanks to the training of the NLU model. It also captures the parameters, or *slots*, if

there are any. In the case of a calendar *App*, when the user says “Add an event anniversary for the 13th November” the request is transformed by the NLU into the *intent* “addEvent” with *slots* for the name of the event “anniversary” and a date “13th November”. The Snips team have already created certain standard *slots*, such as dates, to enable their detection.
Listing 1 shows what the data sent by the NLU to the action code looks like, in the JSON format.



Your assistant needs trial data to train its neural network. You can create your own examples with the aid of the console. To improve the performance of the NLU, you can identify the position and nature of the *slots* in a sentence by highlighting them with the mouse. Moreover, the more numerous and varied your examples are, the better will be the performance of your assistant. The Snips console also offers you a paid service for producing training data.

Don’t forget to save, the console also updates your assistant. You can use the test interface at the right of the console to observe the responses of your assistant to oral or written commands. You can now install your assistant on your RPi, using SAM. The command `sam login` lets you connect to your account used in the console. You can then run `sam install assistant` and select your assistant to install it.

Action

The action code is composed of *callbacks* which are executed when the *intent* which corresponds to them is detected. You can program in any language, however Python and Javascript have a Hermes library available, which simplifies the internal communications with the Snips platform. These two languages are recommended (but not obligatory). You can run your code manually, but the *snips-skill-server* can do it automatically when the RPi starts. For that, create a directory with your choice of name in `/var/lib/snips/skills/`, and put your script in that directory. Your script must have a name beginning with action- and be executable. its role is to listen for *intents*, via an infinite loop. It is possible to link an *App* to a Github directory, and let SAM copy your files to a proper place during the installation of the assistant. It must be public, you can also add it to the *Actions* tab in the console. You can also, in addition to your action file, put in a setup.sh file which will be executed on installation, for example to compile your project.

Example of an assistant

Let’s look at the construction of a practical assistant. This assistant will get you to revise your multiplication tables via a game of questions-replies. From the console, create a new assistant in English called “FirstAssistant”which has an *App* called “TimesTables”. [Note the editing: you will fnd in the App Store the *App* “TimesTablesEN” which is a bit more advanced than the one described here, **Figure 5**]. The player must be

Listing 1. Data Sent by the NLU to the Action Code (JSON Format).

```
“input": "add an anniversary for the thirteenth of
november",
"intent": {
"intentName": "userName:addEvent",
“probability": 0.9662896
},
“slots": [
{
“rawValue": "anniversary",
“value": {
“kind": "Custom",
“value": "anniversary"
},
“range": {
“start": 10,
“end": 22
},
“entity": "common names",
“slotName": "summary"
},
{
“rawValue": "for the thirteenth of november",
“value": {
“kind": "InstantTime",
“value": "2019-11-13 00:00:00 +00:00",
“grain": "Day",
“precision": "Exact"
},
“range": {
“start": 23,
“end": 46
},
“entity": "snips/datetime",
“slotName": "start_datetime"
```

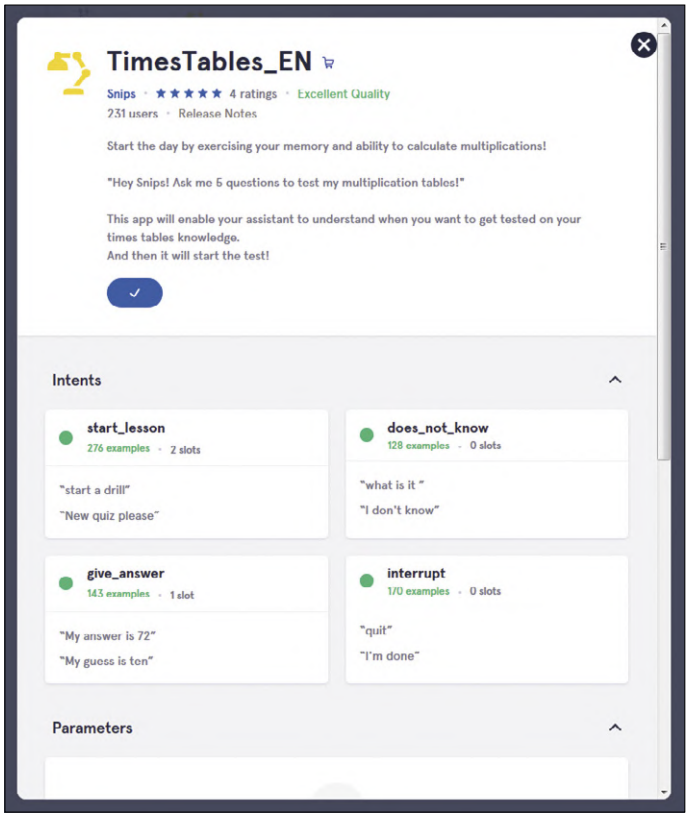


Figure 5: The *TimesTables* app is available online. Select it and then click on *Add 1 App* at the bottom right of the window to add it.

able to start the game, then respond to questions or pass if he doesn't have a reply. We have there our three *intents* named `start_quiz`, `gives_answer` and `doesnt_know`. Note: the action code receives the intent in the format *userName:intentName* (for example `snips:gives_answer`).

- *intent* `start_quiz`: is limited to starting the game, and thus does not need a slot to adjust the behaviour of your assistant. It must include phrases of the type "let's start the game" or "I want to play". It is important to add numerous and varied examples of phrases to make the *intent* robust, so let's add "I feel like playing the game" or "let's play" to the examples before saving (**Figure 6**).

- *intent* `gives_answer`: must allow understanding of a reply which is variable. A *slot* is thus required, we can call it `answer` and give it the predefined type `number`. In the example phrases, don't forget to highlight the numbers if they are not included automatically (**Figure 7**). For this, select the number with the mouse, then click on *Select a slot* then on *answer*. Responses like "forty-nine" or "I think it's twelve" are expected. Data in the JSON format received by your action code will carry the value of the number spoken by the user in its *slots*.

- *intent* `doesnt_know`: follows the same logic as `start_quiz` in that it does not need a parameter. You only need to give sufficient examples to train the models of your assis-

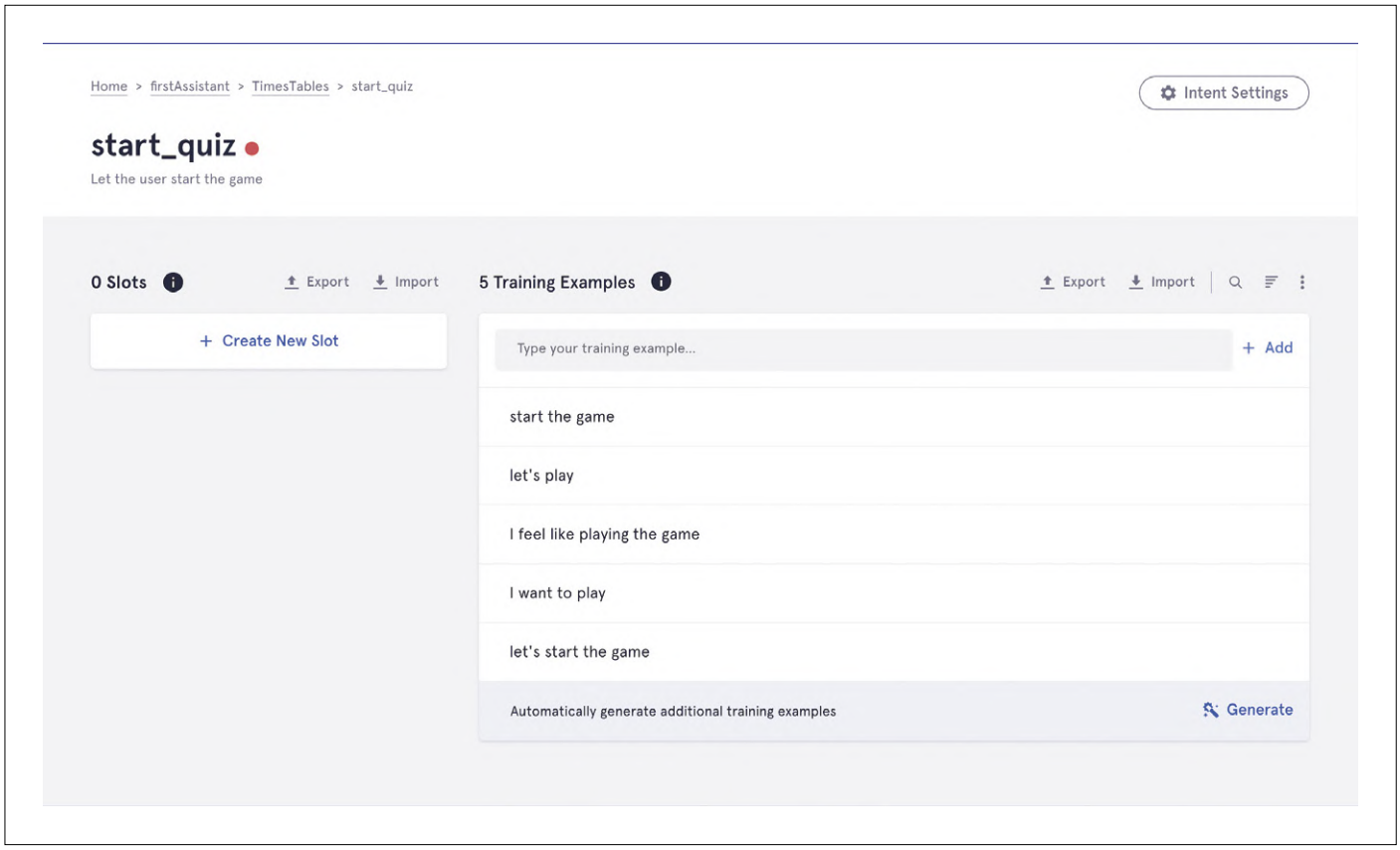


Figure 6: Possible phrases registered in the *intent* `start_quiz` to start a game. This can of course be done in any of the available languages.

tant. “I don’t know”, “I have no idea” or “I can’t get the answer” are pertinent examples.

The assistant is ready to install with SAM, but before this, we need to write the action code in the *Actions* tab.
Here we have chosen a script written in Python, called `action-timestable.py` (downloadable at [2]), and we put it in the directory `/var/lib/snips/skills/timestables/`.

Action code

To run the action code, we use the library `hermes_python`. The first step is to import it, as well as the library `random` (to produce the numbers for the questions).

Definitions

```
#!usr/bin/env python2
# coding: utf-8

from hermes_python.hermes import Hermes
import random
```

The combination “#!” (1st line) is needed so that the *snips-skill-server* can execute the code of your assistant.
Now we define the global variables for our action.

```
MQTT_HOST = "localhost"
MQTT_PORT = "1883" # Standard MQTT port
MQTT_URL = "{}:{}".format(MQTT_HOST, MQTT_PORT) # ==>
   "localhost:1883"

SNIPS_USER_NAME = "snips"
INTENT_STARTS_QUIZ = "{}:start_quiz".format(SNIPS_
   USER_NAME) # ==> "snips:start_quiz"
INTENT_GIVES_ANSWER = "{}:gives_answer".format(SNIPS_
   USER_NAME) # ==> "snips:gives_answer"
INTENT_DOESNT_KNOW = "{}:doesnt_know".format(SNIPS_
   USER_NAME) # ==> "snips:doesnt_know"

sessions_states = {}
```

The three `MQTT_...` variables define the access point to our MQTT server, in our case we connect to *localhost*, because the code is run on the RPi which is running Snips.
The three `INTENT_...`variables define the names of your *intents* prefixed by the name of the user who created the assistant at [1]. These should be replaced by your user name.
The `sessions_states` lets the assistant remember the question posed to the user.

Start the game

Now we define the function to call when the user asks the voice assistant to make him revise the multiplication tables.

```
def starts_quiz(hermes, intent_message):
session_id = intent_message.session_id
```

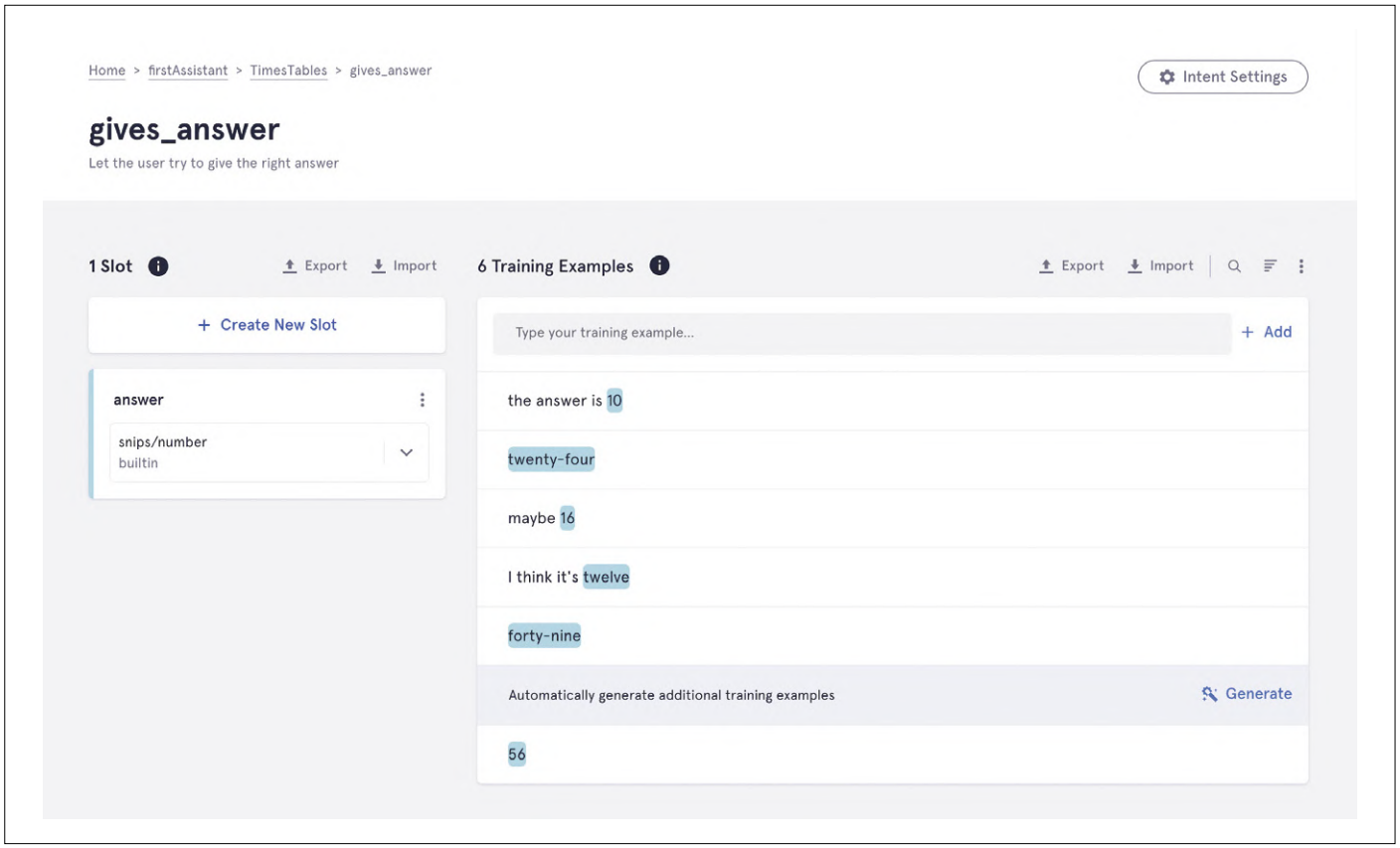


Figure 7: For the *intent* `gives_answer`, the numbers are highlighted for the *Answer* type *slot*.

```
# Generating question
a = random.random() * 10 + 1
b = random.random() * 10 + 1
tts = "What's {} times {} ?".format(a, b)
sessions_states[session_id] = [a, b]

# Define what the assistant is able to understand
intent_filter = [INTENT_DOESNT_KNOW,
   INTENT_GIVES_ANSWER]

hermes.publish_continue_session(session_id, tts,
   intent_filter)
```

Each function associated with an *intent* receives two objects as parameters: `hermes` and `intent_message`.

The first step in constructing a dialogue with Snips is to recover the "dialogue session identifier (`session_id`). This identifier allows us to identify the dialogue that is running in a unique manner; This is important, because several people can use your assistant at the same time (using satellites). A dialogue session starts when the user says the *wakeword* and ends when your code sends an `end_session`.
Then we create a random question in the tables from 1 to 10 and save the question in our `sessions_states` dictionary. This allows us to verify the user's response..
Finally we need to publish an event to continue the dialogue by responding to the user and listening for his response. For this, we run `hermes.publish_continue_session(...)` with the first parameter identifying the session to continue. The second parameter is the phrase/question which the assistant is going to pose to the user (tts = text to speech). The last parameter is the intent filter which allows the assistant to know if the user will respond with a phrase contained in one of the two selected *intents*.

Reply to the question

Here the two *intents* are `INTENT_DOESNT_KNOW` and `INTENT_GIVES_ANSWER` which means that the user can only *give a response* **or** say that he doesn't know.
Now we have to define the function to call when the user gives a response.

```
def gives_answer(hermes, intent_message):
session_id = intent_message.session_id

# get answer from user
answer = intent_message.slot.get("answer")
answer = None if answer is None else answer.first().
   value

# defines TTS
tts_good = "Good answer !"
tts_bad = "Wrong answer ..."
tts_no_answer = "I'm sorry I didn't get your answer"
tts = ""

# Check user's answer
if answer is None:
tts = tts_no_answer
elif answer == sessions_states[session_id][0] *
   sessions_states[session_id][1]:
tts = tts_good
else:
tts = tts_bad

hermes.publish_end_session(session_id, tts)
```

The two initialisation lines for the variable `answer` allow us to recover the value (in the form of a complete number) of the user's response. the second line verifies that the user has given a good response before providing the value.
The lines `tts_... =` define the phrases with which the voice assistant will reply in the three different cases:

1. the user gives a good response;
2. the user gives a bad response;
3. the response from the user was not understood by the assistant.

The `if` lets us determine which of the three cases is correct. Finally we publish the `end_session` event with the `session_id` as the parameter as well as the phrase that the assistant will say before closing this dialogue session.

The function called when the user says he does not know the answer is equally simple. We just publish an `end_session` to close the dialogue, with a predefined TTS.

```
def doesnt_know(hermes, intent_message):
session_id = intent_message.session_id
tts = "Ok, at least try next time !"

hermes.publish_end_session(session_id, tts)
```

We're done

The following function is called once a session has closed after a call to hermes.publish_end_session(...).

```
def session_ended(hermes, session_ended_message):
del sessions_states[session_ended_message.session_id]
```

Here we suppress the dictionary element which was associated with the session which has just closed.

The arrival of Hermes

The last step is to connect all this together.

```
with Hermes(MQTT_URL) as h:

# Subscribe to intents
h.subscribe_intent(INTENT_STARTS_QUIZ, starts_quiz)\
.subscribe_intent(INTENT_GIVES_ANSWER, gives_answer)\
.subscribe_intent(INTENT_DOESNT_KNOW, doesnt_know)

# Subscribe endSession
h.subscribe_session_ended(session_ended)
```

```
# Start Listening
h.loop_forever()
```

The first line creates an instance of the object `Hermes` in a context manager. Note that its argument `MQTT_URL` is the url defined at the start of the file.
The three lines after `# Subscribe to intents` define that for a given *intent* (defined by our global variables), a given function is associated. It is thanks to these three lines that Snips knows which function to call for each of the three *intents* received.
Then in the same manner we define that the function session_ ended will be called when a dialogue session finishes.
Finally we execute the function which listens for the different Snips events and calls the functions associated with each *intent*.
To summarise, the action code has the following structure:

- importing the libraries;
- definition of variables;
- definition of actions for each *intent;*
- initialisation of a context (*Hermes*).

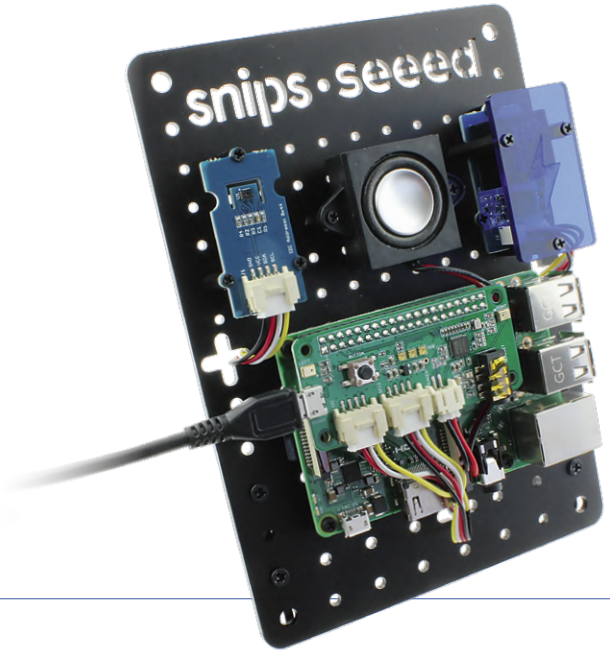
We hope that this first article about the Snips voice assistant will get you on the way to using it. You are now ready to add voice commands to any of your Raspberry Pi projects. To make this easier, don't forget to check in the *App Store* in case someone has already designed an *App* close to your needs.

(180733-02)

@ WWW.ELEKTOR.COM

→ **Snips Voice Interaction for Raspberry Pi, base kit**
www.elektor.com/snips-voice-interaction-base-kit
See also online review [3]

→ **Raspberry Pi 3B+**
www.elektor.com/rpi-3b-plus



Web Links

[1] Snips console, tools and documentation : https://console.snips.ai

[2] Python Code of the App on the Article Page: www.elektormagazine.com/180733-01

[3] Review: Snips — Speech Recognition for the Raspberry Pi: www.elektormagazine.com/news/review-snips-speech-recognition-for-the-raspberry-pi

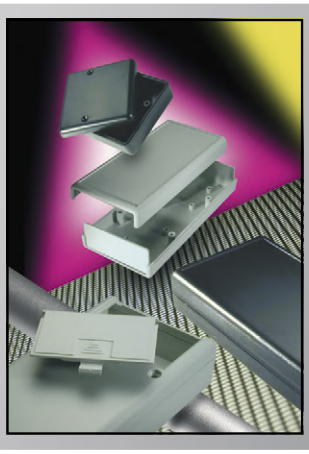
Advertisement

HAMMOND MANUFACTURING®

Plastic enclosures standard and miniature

More than 5000 different enclosure styles:
hammfg.com/electronics/small-case

01256 812812
sales@hammond-electronics.co.uk



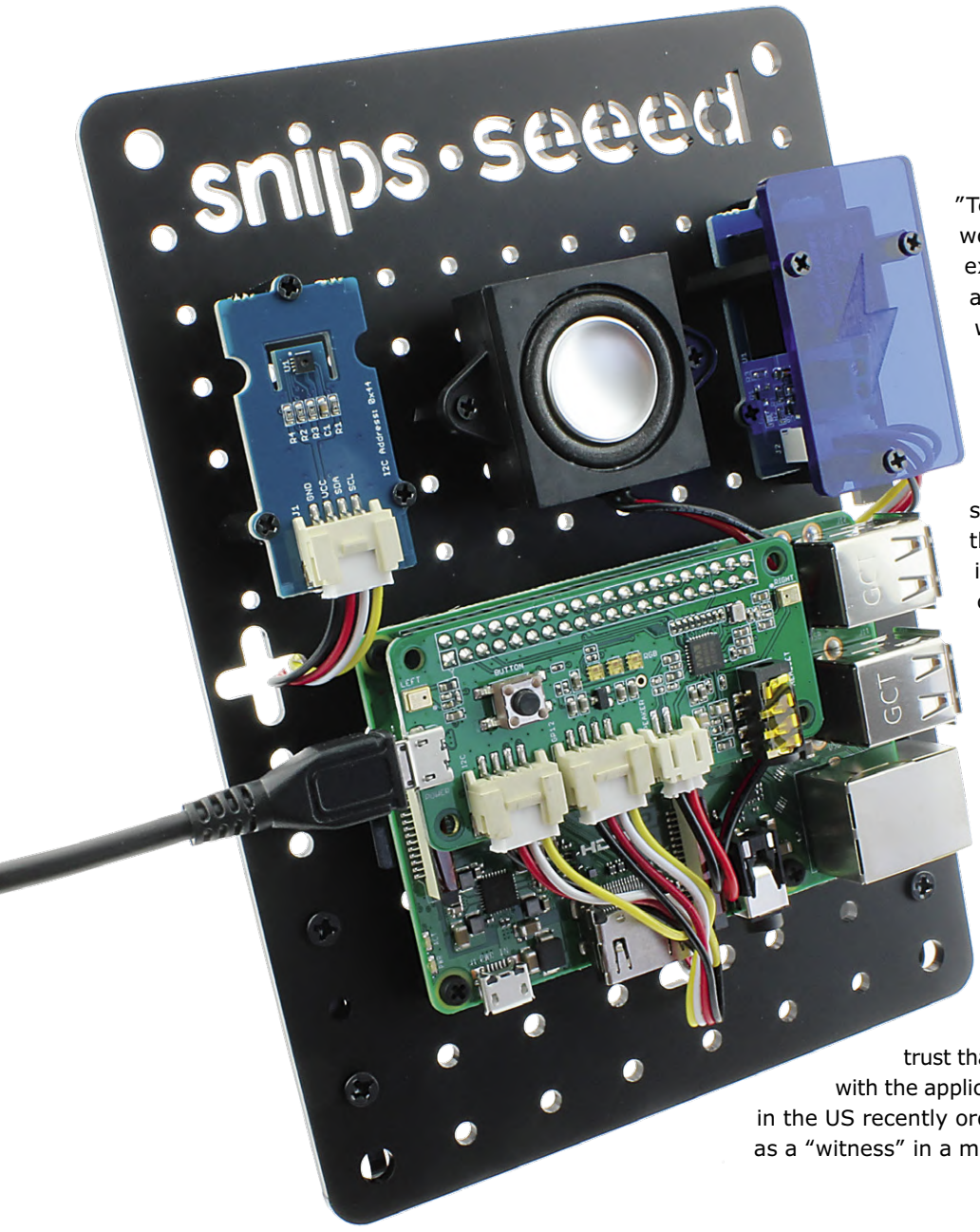


Snips Speech Recognition

with an example for the Raspberry Pi

By **Mathias Claußen** (Elektor Labs)

Language assistants like Alexa, Google Home or Siri are now commonplace. However, the actual recognition of the spoken words is not done locally, but in the Cloud of the provider – which raises questions about data privacy. There are other solutions such as Snips – a speech recognition system that runs entirely on local hardware like the RPi. We took a look at a speech recognition kit and show how you can use it to develop your own applications...



"Tea, Earl Grey – hot" – 10 years ago it would have been pure science fiction to expect a computer to respond appropriately to that sort of spoken request. That was before the arrival of Alexa, Google Home and Siri. The language assistants of Amazon, Google or Apple can listen to everything we say and respond (not always as expected). Anyone who wants to write their own application using speech recognition will quickly realize that the home assistant is not quite what it seems to be. All of these systems are continually looking for a match between incoming sound samples and a limited list of keyword templates. After we have enabled the wizard with the keyword, everything that it collects from audio signals is sent directly to the respective vendor's servers. The spoken-word recognition process does not take place locally but somewhere in the provider's cloud. In this context a cloud is nothing more than a collection of computer system resources that exists somewhere in a data centre. What eventually happens to that data, we as the user and app developer cannot see. We can only trust that the provider deals with it in accordance with the applicable data protection regulations. A judge in the US recently ordered Amazon's Alexa to be summoned as a "witness" in a murder case.

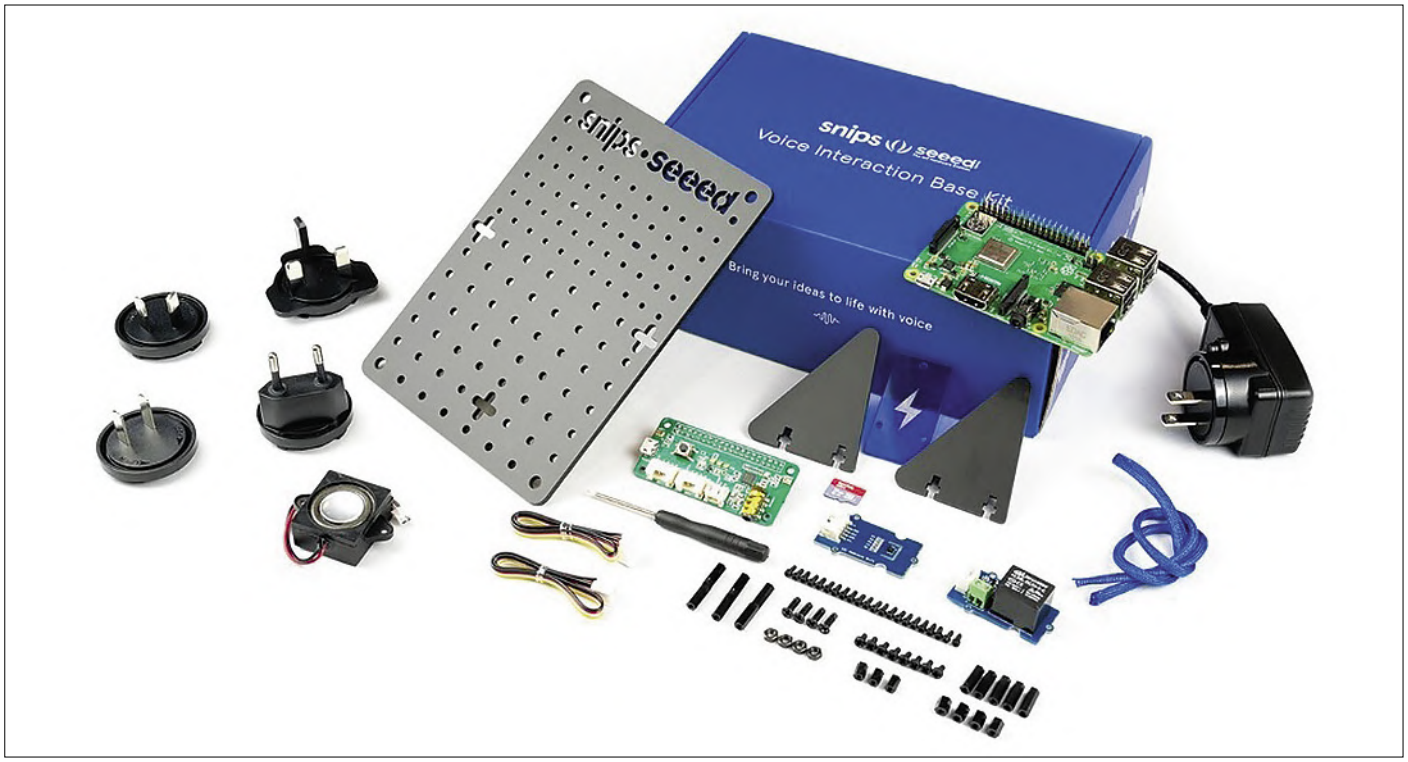


Figure 1: The kit after unpacking.

Anyone with a fear of Big Brother will be happy to know there are alternatives available such as Snips — this is speech recognition software that works completely offline and ensures your data does not go any further than your home network. It doesn't need a permanent Internet connection or a bank of computers to perform speech recognition. To explore the system further we will use the Snips Voice Interaction Kit which uses a Raspberry Pi 3B+ as the base unit.

In addition to the RPi we need a few more components for this speech recognition application. Let's take a look what's in the kit (**Figure 1**).

The kit

The kit arrives packed in a plain blue cardboard box. Inside are all the components including the Raspberry Pi 3B+, a 5-V 3-A mains power adapter, a relay, a temperature and humidity sensor, a speaker, a ReSpeaker 2-Mics Pi HAT, a mounting plate and an SD card with pre-installed system along with a quick start guide.
This gives us all the ingredientsto build our own small language assistant. When assembling the kit, use the appropriate spacers and screws. Unfortunately the manual skips over some details here so we are left wondering which screws go where. After some trial and error we finally get it sorted. The RPi board is not ideally positioned to easily access the SD card slot.

The perforated acrylic mounting board

An acrylic mounting board is supplied onto which you can fix the various modules using self-tapping screws. The matrix of fixing holes allows a certain degree of freedom as to where the modules can be fixed.

The sensors and actuators

The temperature and humidity sensor is a Sensirion type SHT31 mounted on a Grove module. The temperature can be measured to ±0.3 °C and the humidity with ±2% accuracy.
The relay is also a Grove module from Seeed Studio. Its contacts can switch 250 VAC / 30 VDC with a current rating of 5 A and should be good for 100,000 switching cycles. Although the relay can switch mains voltages the board has not been designed to operate with them because there is no insulation provided and it would be easy to accidentally touch live conductors. As long as the load voltage is below the extra-low voltage safety level (<25 VAC / <30 VDC) and within the relay parameters, you can safely use this relay module for experimentation.
A small speaker is included so you can hear what the language assistant is saying. As you would expect from a small speaker the sound quality is quite poor but adequate for the purpose.

ReSpeaker 2-Mics Pi HAT

The heart of the system — apart from the RPi itself — is the ReSpeaker 2-Mics Pi HAT [1]. This is built around a WM8960 audio codec. Unfortunately, this one is not yet supported directly by Raspbian, and the driver also has a few bugs. Therefore its necessary to when using the shield with a naked Raspbian-image, the driver itself has to be translated for the RPi. Here are the instructions to manually configure the driver. The WM8960 integrates a Class D power amplifier which can supply 1 W into 8 Ω per channel and also includes wiring for microphones. Two microphones are installed on the board; the connections for a speaker or headphones are also available.
On the board there are also three RGB LEDs (APA102), which connect via SPI to the Raspberry Pi. These can signal when the voice assistant starts to listen (or has not understood).

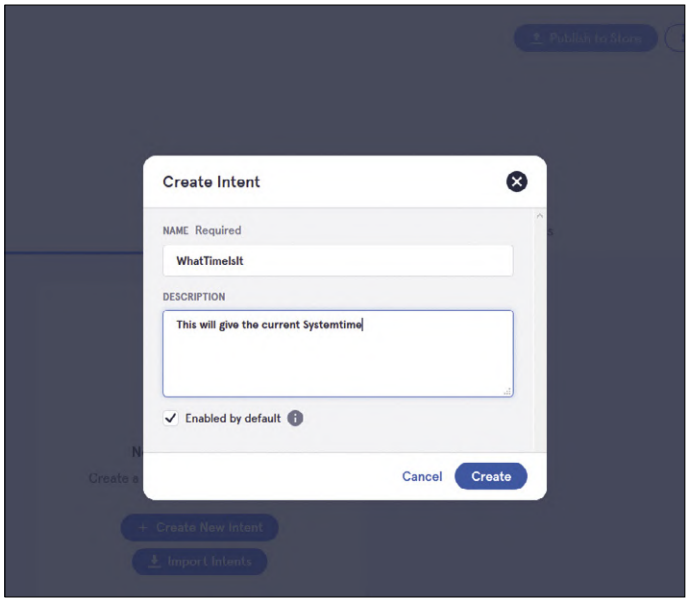


Figure 2: Creating an Intent (speech command).

A pushbutton is also provided to provide interaction with the system. Finally, there are the connections for the relay, the temperature and humidity sensor.
A micro-USB port on one side provides convenient power supply connection to the RPi. Unfortunately, the header connector strip is not feedthrough so we can't stack another HAT on top and it's not possible to use a SPI-TFT display as a GUI.
Make sure you connect the cables to the correct ports when hooking up the relay, humidity and temperature sensor. After everything has been assembled and the SD card inserted we can begin. An HDMI monitor and a USB keyboard are useful now so that we can track the RPi's first boot process.

Listing 1. State output and version number of the individual Snips components.

```
Connected to device snips-base.local
OS version ........... Raspbian GNU/Linux 9 (stretch)
Installed assistant .. MakerKitBundle_EN
Language ............. en
Hotword .............. hey_snips
ASR engine ........... snips
Status ............... Live

Service status:

snips-analytics ...... 0.60.8 (running)
snips-asr ............ 0.60.8 (running)
snips-audio-server ... 0.60.8 (running)
snips-dialogue ....... 0.60.8 (running)
snips-hotword ........ 0.60.8 (running)
snips-nlu ............ 0.60.8 (running)
snips-skill-server ... 0.60.8 (running)
snips-tts ............ 0.60.8 (running)
```

After booting, the system is ready for its first test. Try saying "Hey Snips - What's the temperature?" The system should reply with the current temperature, then "Hey Snips — Turn the relay on" to switch the relay on. These commands should work straight out of the box without the need to make any configuration changes. Now we get to the more interesting part when we begin to implement our own applications/commands using speech recognition.

Setup

The Raspberry Pi now needs access to our network, the easiest way to do this is by using a cable. If you want to do the whole thing via Wi-Fi, the SD card must be taken out of the RPi and the file `wpa_supplicant.conf` placed in the drive with the name boot. The file contains:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=<your_country_id>
network={
ssid="<your_ssid>"
psk="<your_password>"
}
```

The corresponding country, SSID and password entries must match your own Wi-Fi credentials.
Using snips-base.local you can connect to the Raspberry Pi via SSH and explore the system. The snips software is already preinstalled on the system.
After the basic setup of the RPi, which is quite straightforward, we install Sam. Sam is a command-line interface that lets you monitor and manage the Snips software on the Raspberry Pi. To install Sam we need to open a terminal and enter sudo npm install -g snips-sam. We can now link to the Snips software on the Raspberry Pi with sam connect snips-base.local. With sam status we can then output the state and the version numbers of the individual components. The output should look like **Listing 1**.
Since the software is still subject to updates, it can occur that one or other of the versions will sometimes not work properly. To get the latest version, we now run `sam update`. The RPi now downloads and installs the latest version. Finally we need to run the `sam update-assistant`. This will then also update the applications and all Snips components should be online again. If we now enter `sam status`, all components (except `snips-analytics`) now run and the RPi should respond to the example commands from the tutorial.

Build your own application

After creating an account [2] you can use the web interface to create your own application, and even better, train the assistant on your own voice commands. As a small demo, we will get the language assistant to give us the current time.
We first create a new wizard, in this case we call it WhatTimeItIs. Within a wizard, apps can now be created. We create an app called "currentTime". This app is then responsible for issuing the current time on command.
In order for the app to understand a voice command, we have to train it as an "intent" (short for intention), see **Figure 2**.
There are a couple of built-in modules that can make things easier for us if we Talking numbers or a date. Since our key-

word questions are not available as a finished module, we need to create them. We want to listen to “Time”, as well as a few synonyms.
Now that we have finished learning, we can take care of the action that our voice command should trigger. For this we go to the point Actions and choose for our demo *Code snippets*. Currently we can write Python3 and Python2 code via the web interface and have it executed as an “action”. (**Figure 3**).
For more detail consult the snips documentation [3].
To output the time, we use the Python3 code in **Listing 2**. This code will be executed when the word “Time” or one of its synonyms is detected. In response, we build a string, which is then sent to the text-to-speech software module which reads out the time for us.
With this we have created our first application and can test it. For this we still have to transfer them to the Raspberry Pi. In our attached Linux, we enter `sam install assistant` and install the newly created wizard. Then we can ask for the time with “Hey Snips, what’s the time”.

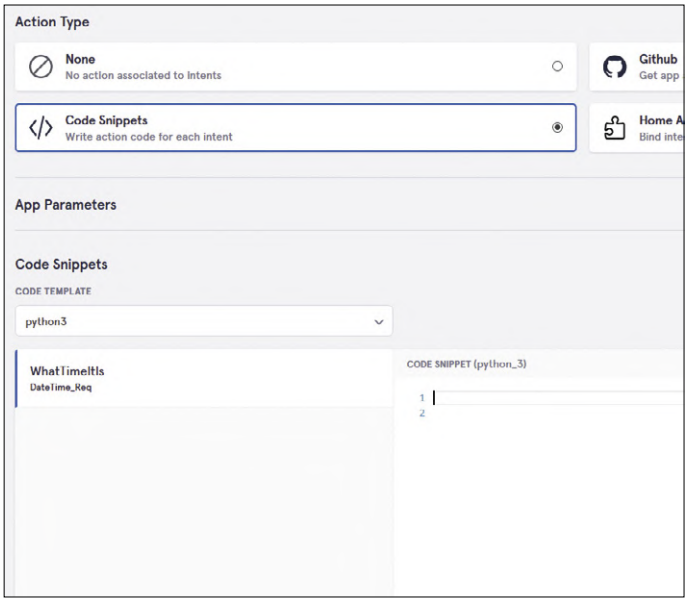


Figure 3: Entering code for command execution.

To sum up

The kit contains everything you need to start experimenting with voice control including speakers, microphones, sensors and relays. Unfortunately, the output signals from the two built-in microphones is poor quality and quite noisy. This does not affect the speech recognition but anyone following the instructions to record a new keyword to activate the speech assistant will unfortunately need a better quality microphone. When assembling the hardware you need to pay attention, one or two additional pages in the manual with illustrations showing the spacer placing would have been helpful.
It is not ideal that the wm8960 codec is not currently supported by the current Pi kernal builds making it necessary to manual build to run the sound card in the official Raspbian. If you want to create your own image based on Raspbian, then you will need to compile and patch the driver yourself. If a new version of Snips is installed the assistant will need to be updated again or reinstalled.

Once you get used to using the Snips.ai web interface, it’s fun to work with the system and explore the possibilities the system offers. Above all, it’s reassuring to know that at the end of the day, I am the only one with access to all my data. ⏮

(180208-02)

@ WWW.ELEKTOR.COM
→ Snips Voice Interaction Base Kit
www.elektor.com/snips-voice-interaction-base-kit

Listing 2. Python code for our application.

```
import time
now = time.localtime()
if len(intentMessage.slots.DateTime_Req) > 0:
    date_or_time = intentMessage.slots.DateTime_Req.first().value # We extract the value from the slot
    result_sentence = "Current time is : {} {} ".format(str(now.tm_hour), str(now.tm_min))
                                        # The response that will be said out loud by the TTS engine.
else:
    result_sentence = "Time is running out"
current_session_id = intentMessage.session_id
hermes.publish_end_session(current_session_id, result_sentence)
```

Web Links

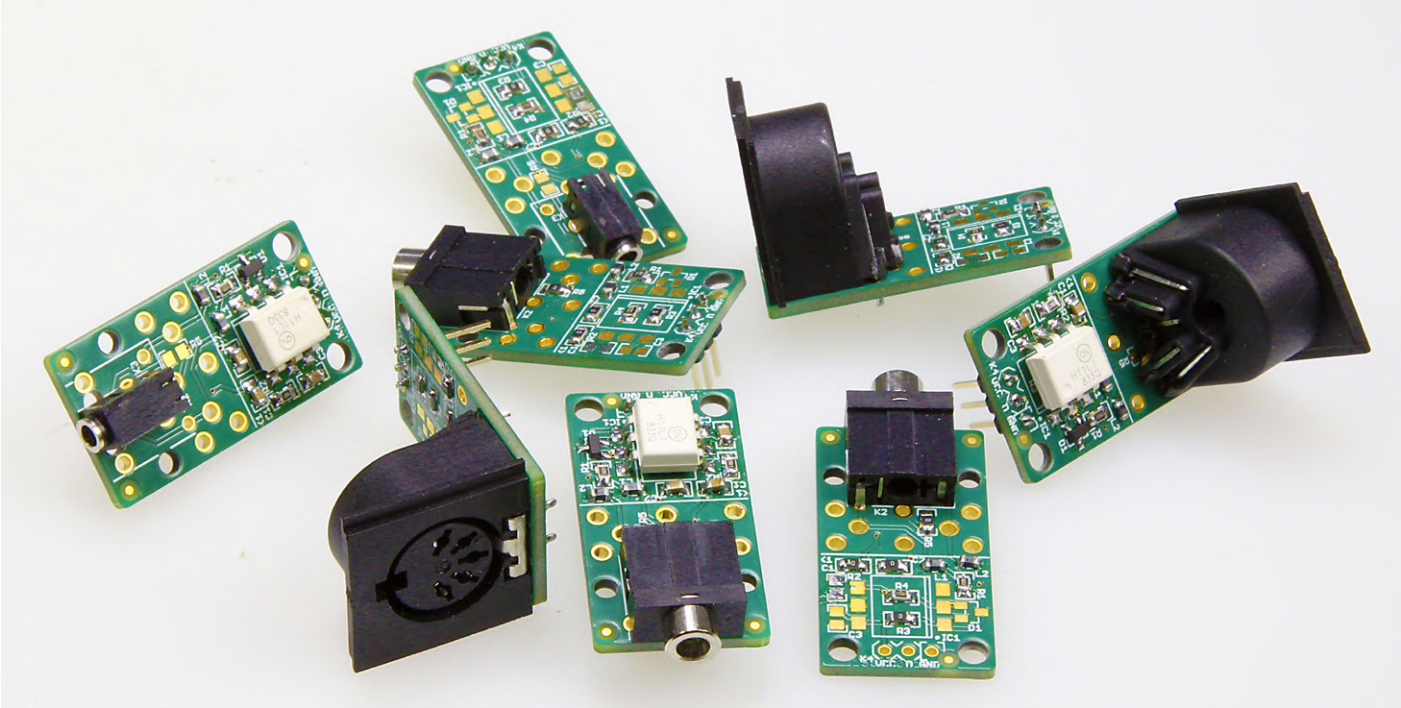
[1] Embedding an audio driver: http://wiki.seeedstudio.com/ReSpeaker_2_Mics_Pi_HAT/
[2] Snips homepage: https://snips.ai/
[3] Snips documentation: https://docs.snips.ai/articles/console/actions/actions/snippets

eLABS PROJECT

MIDI I/O Breakout Board

supports DIN and TRS connectors

By **Clemens Valens** (Elektor Labs)



At the start of 2019, some 35 years after its introduction, version 2.0 of the Musical Instrument Digital Interface, better known as MIDI, was announced. MIDI 2.0 introduces many new features, two of them being two new connectors. A new universal MIDI interface breakout board is due and here it is.

Quick Features

- One PCB for MIDI In, MIDI Out and MIDI Thru
- Configurable for 5-V and 3.3-V systems
- Supports DIN, 3.5-mm TRS and 2.5-mm TRS connectors
- Compliant with latest MIDI standard

At the time of writing this article (April 2019) the MIDI 2.0 specification has not been released yet. We will therefore not go into any detail here. What we do know is that the new standard will remain 100% compatible with MIDI 1.0.

TRS? Like "jack"?

To be honest, the new MIDI connector specification was released in the summer of 2018, and therefore is pre-MIDI 2.0. In a *Letter of Agreement for Recommended Practice* the Tip-Ring-Sleeve (TRS) socket/plug combination got accepted as the MIDI connector/plug, and the way to wire it to the traditional 5-way DIN connector/plug is standardized. Most people will know the TRS connectors as 'audio jack' plugs and sockets. Actually, the new connectors are just one but with two different diameters: 3.5 mm and 2.5 mm, where the latter is the recommended type. The motivation for this addition to the standard is simple and size related: the venerable DIN plug is too bulky to fit in modern, flat smartphone-like equipment.

New, new, all new!

The new MIDI breakout board (BoB) we present here can be equipped with either one of the three connector types: DIN, TRS-3.5 or TRS-2.5. Furthermore, the BoB supports RF grounding and 3.3-volt signalling, and the same PCB can be configured as either a MIDI In or MIDI Out (or MIDI Thru, which is the same thing as Out).

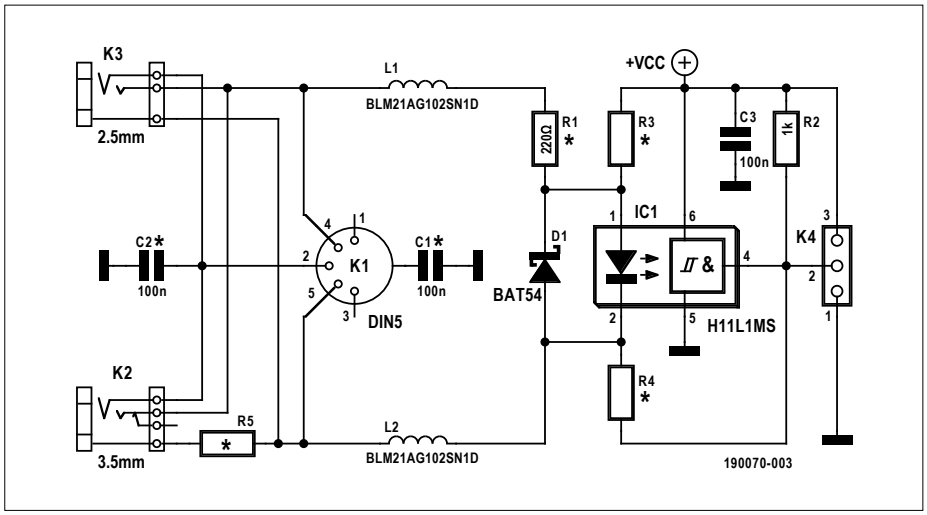


Figure 1: This slightly muddled schematic of the MIDI I/O BoB is what you get when you combine multiple circuits into one configurable circuit.

PROJECT DECODER

MIDI
TRS DIN
Breakout Board

entry level
intermediate level
expert level

1 hour approx.

soldering iron

£12 / €15 / $17 approx.

The schematic of the MIDI I/O BoB is shown in **Figure 1**. Due to configuration options and EMI/EMC and RF measures, there are a few more components than in a typical MIDI interface design.

Ground loops *versus* **EMC**

The original MIDI 1.0 specification stated that in order to avoid ground loops between devices "*The grounding shield connector on the MIDI jacks should not be connected to any circuit or chassis ground.*" However, in 2014 this was relaxed and an optional connection between the shields of MIDI Out and MIDI Thru connectors to Ground was allowed to improve EMI/EMC performance. MIDI In connector shields may be grounded through a small capacitor, but never directly. Our MIDI I/O BoB therefore allows for these grounding options using C1 and C2. When the board is used as a MIDI In, C1 and C2 are either left off, or mounted as 100-nF (US: .1 µF) capacitors; in the case of a MIDI Out (or Thru) C1 and C2 can be 0-Ω resistors.

COMPONENT LIST

MIDI In

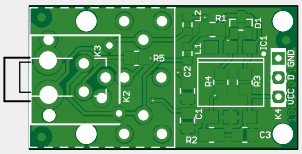
Resistors
All SMD 0805
R1 = 220Ω
R2 = 1kΩ
R5* = 0Ω

Capacitors
All SMD 0805
C1,C2,C3 = 100nF

Inductors
All SMD 0805
L1,L2 = ferrite bead, 1kΩ @ 100MHz

Semiconductors
D1 = BAT54
IC1 = H11L1 or equivalent

Miscellaneous
K1* = 5-way DIN socket, 180° style
K2* = 3-way TRS socket, 3.5mm diameter
K3* = 3-way TRS socket, 2.5mm diameter
K4 = 3-pin pinheader, 0.1" pitch
PCB # 190070-1 from Elektor Store



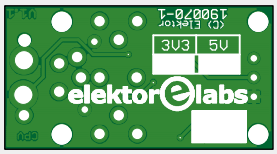
MIDI Out / MIDI Thru, 5 V

Resistors
All SMD 0805
R1,R4 = 220Ω
R3,R5* = 0Ω

Capacitors
All SMD 0805
C1,C2 = 0Ω*
C3 = 100nF

Inductors
All SMD 0805
L1,L2 = ferrite bead 1kΩ @ 100MHz

Miscellaneous
K1* = 5-way DIN socket, 180° style
K2* = 3-way TRS socket, 3.5mm diameter
K3* = 3-way TRS socket, 2.5mm diameter
K4 = 3-pin pinheader, 0.1" pitch
PCB # 190070-1, from Elektor Store



MIDI Out / MIDI Thru, 3.3 V

Resistors
All SMD 0805
R1 = 33Ω, 0.5W
R3,R5* = 0Ω
R4 = 10Ω, 0.25W

Capacitors
All SMD 0805
C1,C2 = 0Ω*
C3 = 100nF

Inductors
All SMD 0805
L1,L2 = ferrite bead 1kΩ @ 100MHz

Miscellaneous
K1* = 5-way DIN socket, 180° style
K2* = 3-way TRS socket, 3.5mm diameter
K3* = 3-way TRS socket, 2.5mm diameter
K4 = 3-pin pinheader, 0.1" pitch
PCB # 190070-1, from Elektor Store

Avoid RF interference

Ferrite beads L1 and L2 attenuate RF interference but may be replaced by 0-Ω resistors for simplicity (depending on what you have in stock, for instance). R3 and R4 are configuration resistors and are only required for MIDI Out (and Thru) applications. To prevent anyone from accidentally mounting them on a MIDI In module, they are placed in the same position as optocoupler IC1.

MIDI In calls for an optocoupler

The optocoupler is a fast device ensuring good signal quality and low latency between In and Thru in the case of chaining multiple MIDI devices. Diode D1 protects it against reverse-polarity signals.

R2 is the output resistor for the open-collector output on IC1. You can lower its value to improve speed slightly, but at the expense of a higher current consumption. IC1's minimum supply voltage is 3 V, which sets the lower boundary for V_{CC} for the MIDI In configuration (the upper extreme for IC1 is no less than 16 V).

C3 provides supply bypassing and is probably more of a *good practice* thing than really necessary, but hey, you never know what lengths of wires or traces people connect to K4 (keeping them as short as possible is also *good practice*).

Resistor R5, a 0-Ω resistor, has no function other than to provide a way to share a shield pin of K2 with a shield pin of K1 without unintendedly connecting everything to GND. It must only be mounted when K2 is mounted also.

Configuring for MIDI Out and MIDI Thru

A MIDI Out or MIDI Thru module consumes fewer parts since IC1, D1 and R2 are not needed. For these configurations R3 is a 0 Ω resistor that connects R1 to V_{CC}. R4 is the MIDI output current limiting resistor.

The values of R1 and R4 depend on V_{CC}. For a 5-V supply voltage, both take a value of 220 Ω; this is the traditional MIDI Out port for 5-V systems. However, for 3.3-V systems their values must be adapted as follows.

Calculations for low-voltage signalling

At its introduction the MIDI standard specified a current of 5 mA through the MIDI input's optocoupler LED to make sure that the LED would light well. Modern optocouplers usually don't need that much current (ours only needs 1.6 mA), but since a modern MIDI Out must still be capable of driving an vintage MIDI In, 5 mA is what is needed.

A MIDI input presents itself as a 220-Ω resistor in series with an LED; the MIDI standard assumes that this LED can have a forward voltage as high as 1.9 V. This means that the output signal level V_{TX} must be at least:

$$V_{TX} \geq 0.005 \times 220 + 1.9 = 3.0\ V$$

Short-circuit current limiting is required, which is the function of R4. R1 and R4 together (remember that R3 = 0 Ω) will introduce a voltage drop that has to be overcome by V_{TX} as well. Therefore

$$V_{TX} - 3.0 \geq 0.005 \times (R1+R4)$$

With V_{TX} = 3.3 V, we can calculate

$$R1 + R4 = 0.3 / 0.005 = 60\ \Omega$$

The MIDI Manufacturers Association proposes R1 = 33 Ω and R4 = 10 Ω, a total of 43 Ω, to allow for all sorts of tolerances on resistor values and supply and output voltages.

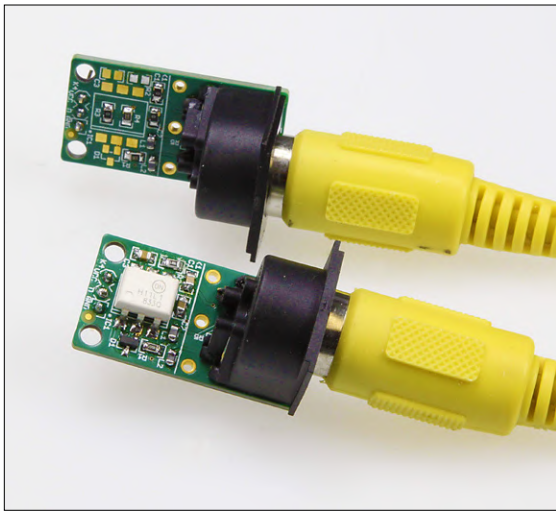
Beware of short-circuits

An easy to overlook detail is the current through R1 and R4 when the remote MIDI input is faulty and for some reason shorted to GND:

$$V_{CC} / R1 = 3.3 / 33 = 0.1\ A$$
$$P_{R1} = (0.1)^2 \times 33 = 330\ mW$$

Because R1 is connected to the transmitter's power supply which we can only assume to be powerful enough to deliver this 100 mA current, R1 must be rated for 0.5 W.

Similar reasoning holds for R4 (a 1.2-W type would be required), except that for this resistor it is assumed to be driven by an open-collector or open-drain output with a pull-up resistor. This resistor with a value of a few hundred ohms (typically) would limit the current, allowing the power rating of R4 to be relaxed to 0.25 W. Therefore, be careful when driving the 3.3-volt MIDI Out from a digital driver. This limitation does not apply to the 5-V MIDI Out.



Finally...

The above description shows that even for simple circuits there are a good many things to e-worry about, especially when trying to interconnect equipment from different origins and eras.

Final note: a complete set of nine component stuffing tables for the MIDI I/O BoB can be found at [1]. ⏮

190070-01

Web Link

[1] MIDI I/O BoB at Elektor Labs: www.elektormagazine.com/labs/midi-io-break-out-board

@ WWW.ELEKTOR.COM

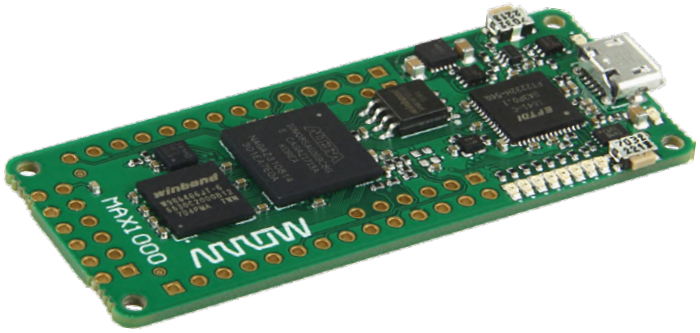
→ MIDI IO Breakout Board, configured as input – Partly Assembled Module
www.elektor.com/190070-91

→ MIDI IO Breakout Board - Bare PCB
www.elektor.com/190070-1

The SCCC Project (3)

Homebrew soft-core processor and C compiler

By **Martin Oßmann** (Germany)



In the previous installment of this series we limited ourselves to running our DIY CPU in simulation. In this part we will return to the MAX1000 board. We will connect this small development board to a PC using a couple of serial interfaces, which will allow us to upload programs and monitor serial input and output. We will also read data from the acceleration sensor that is included on the board.

We already made use of the MAX1000 board in the first part of this series [1]. We will now briefly describe how to get a compiled program into the FPGA. The process of compiling the C program itself is relatively quick, and results in two files, *codemem.txt* and *datamem.txt*. The content of these files must be loaded, in effect, into the CPU's memory space, which is physically located inside the FPGA.

The following lines of the Verilog source file *sCCCPcpu1v01.v*, which defines the CPU, achieve this.

```
parameter sCcodeMemAddressWidth=12 ;
parameter sCdataMemAddressWidth=10 ;
reg [40-1:0] sCcodeMem [0:(1 <<
   sCcodeMemAddressWidth)-1] ;
reg [32-1:0] sCdataMem [0:(1 <<
   sCdataMemAddressWidth)-1] ;

initial begin
  $readmemh("CodeMem.txt",sCcodeMem) ;
  $readmemh("DataMem.txt",sCdataMem) ;
end
```

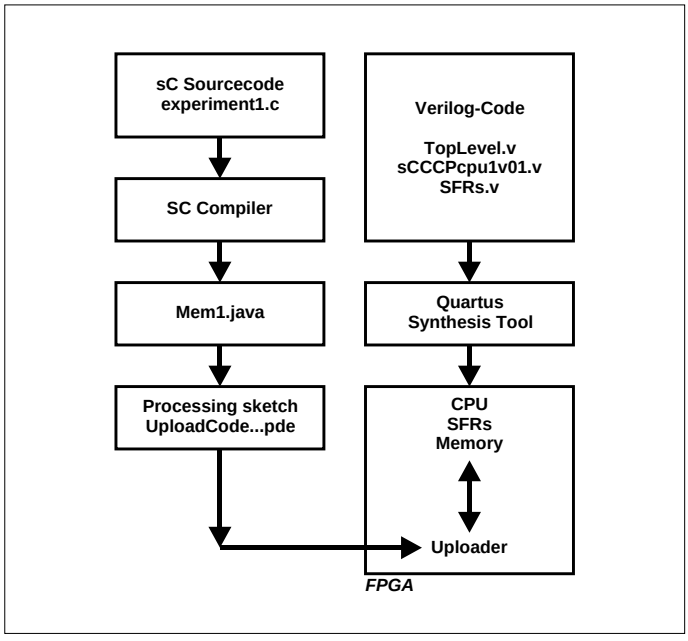


Figure 1. The uploader can be used to send compiled application programs to the CPU after the synthesis step: this makes the development cycle much quicker.

The first four lines above specify the word size and the number of words of storage for the code and data memories.
The lines after 'initial begin' specify that the code and data memories are to be initialized with the contents of the named files. That means that these files must be available at the time the FPGA code is synthesized. We therefore synthesize the FPGA design after compiling the C program to generate the bitstream that is to be loaded into the FPGA. This step is relatively time-consuming (on the author's machine it takes about five minutes) and the synthesis process must be repeated from scratch after each change to the C source code.
This resynthesis process becomes somewhat tedious after a while, and we might wonder if it is possible to load the program and data memory after synthesis has been done. One way to achieve this is to build an 'uploader' into the FPGA design.
Let us look at how this is done. In the working directory of *experiment5* (available for download at [3]) there is a Quartus project file (.qpf extension). In order to use the uploader we need to synthesize this project and load it into the FPGA.

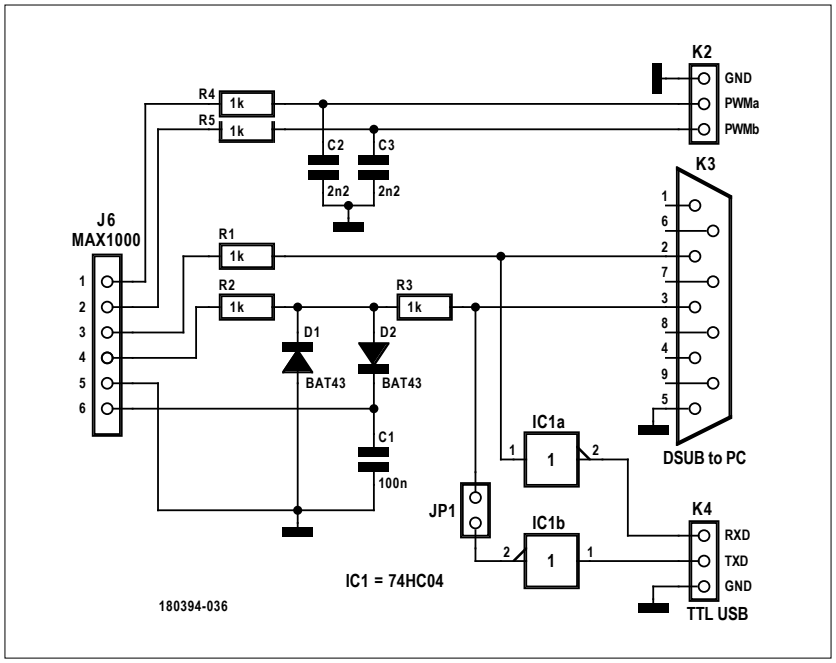


Figure 2. Hardware extension to allow uploading programs over the service interface. For the time being we will not be using the PWM interface.

The workflow that we will be able to use as a result is illustrated in **Figure 1**.
A serial interface will be used to send data and program code to the FPGA. So first a few remarks on how that is implemented.

Our first hardware extension

The serial interface uses the standard 8N1 protocol at 115200 baud. The hardware shown in **Figure 2** is used to connect the PC's serial port to the MAX1000 board. If on the PC side you are lucky enough to have a real serial port, this hardware can be connected to the 9-pin connector. In that case IC1 is not required (and jumper JP1 can be removed). This is the setup used by the author with his serial port. Although the circuit does not strictly meet the interface standards, it nevertheless works well.
Many readers may have a USB-to-TTL serial cable, with the signal levels inverted with respect to the RS-232 levels. Such a cable can be connected at K4. To make things even simpler it is possible to connect the USB-to-TTL cable directly to the FPGA, but in that case the inversion of the signals must additionally be implemented inside the FPGA.

Serial service interface

To upload an application program over the 'service interface' requires a small program running on the development PC. The author implemented this program using Processing [4]. The program can be downloaded as *Processing\UploadCodeViaSerial1v01* at [3], and it can be launched directly from the Processing development environment. Before running it, however, it is necessary to modify the line

```
String serialPort="COM3";
```

to suit the COM port that you will be using for communication with the hardware.

To test whether the service interface is working, send the string

```
1234x
```

to the port, for example using a terminal emulator program. The response should be a row of hexadecimal characters

```
000000001234 000000000000 1234
```

including an echo of the hexadecimal data that were sent.
In the directory *Processing\UploadCodeViaSerial1v01* you should find a further file Mem1.java, assuming you have started the SCCCP development environment supplying the output option *-p* (for 'Processing'). This file will later hold the object code for our application program and will be used by the Processing environment in order to carry out the upload operation.

Connecting peripherals

The CPU that we will use for experiment 5 is equipped with other functional blocks alongside the UART used for the service interface. These include a real-time counter (RTC), a pulsewidth modulator (PWM), an analog to digital converter, a GPIO port, and, last but not least, a user UART. The user UART can be connected to the PMOD2 connector on the MAX1000 board (J6) using the hardware shown in **Figure 3**.

We will now look at how the CPU in the FPGA can be expanded using simple peripheral modules. First we will use three pins as general-purpose I/O (GPIOs). In Verilog we do this as follows.

```
parameter GPIOINPWIDTH = 3 ;
wire[GPIOINPWIDTH-1:0] GPIOinp ;
parameter GPIOOUTPWIDTH = 3 ;
reg[GPIOOUTPWIDTH-1:0] GPIOoutp ;
```

To read data from the expansion module the CPU needs to execute an `INPA c` instruction, where `c` (for 'channel') is the number of the port to which the GPIO peripheral is connected in the CPU implementation in the FPGA.

In Verilog the mechanism is modeled using the variable `inpA`, which holds the value that will eventually end up in register R0 as a result of executing the above instruction. The source of the data for this variable is selected using a large multiplexer, which is implemented as follows.

```
assign inpA=
  (inpAchannel==0) ?  { 32'h12341234   } :
  (inpAchannel==1) ? sCuartRXdata :
  (inpAchannel==2) ? sCTXuartBitsLeft :
  (inpAchannel==3) ?{ 20'h12300 , ADCval[12-1:0] } :
  (inpAchannel==4) ? RTCtimer :
  (inpAchannel==5) ? debugReg1 :
  (inpAchannel==6) ? aDacSignal :
```

```
(inpAchannel==7) ? bDacSignal :
(inpAchannel==18) ? GPIOinp :
32'h12341111 ;
```

For example, if channel 0 is selected then the value '12341234' will always be returned, which can be used for testing; channel 1 returns the data received by the user UART; and channel 18 reads the signals on the GPIO pins, which we will make use of in experiment 5.

Operation in the other direction is similar. If data are output to channel c using the instruction `OUTA c`, the signal `outAstrobe` is activated and the data are routed to the correct destination using a large if-then-else construct. The Verilog code to implement this is as follows.

```
always @(posedge sCclk) begin
  if(outAstrobe) begin
    if( outAchannel==0) begin
      ...
      else if( outAchannel==5) begin
        LEDs<=outA[8-1:0] ;
        end
        ...
      else if( outAchannel==18) begin
        GPIOoutp<=outA[GPIOOUTPWIDTH-1:0] ;
        end
      ...
    end
  end
```

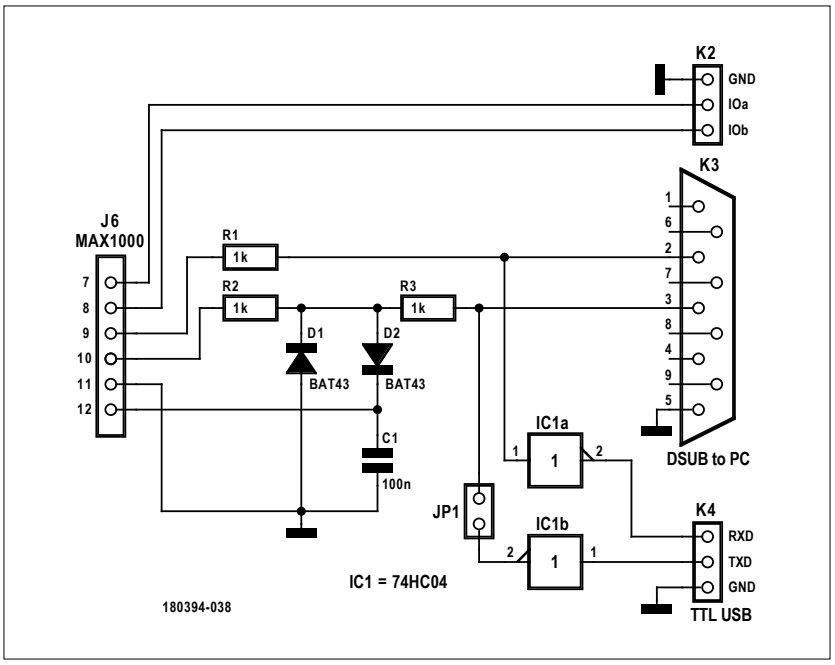


Figure 3. This extension lets us carry out simple I/O experiments.

From this code we can see, for example, that the LEDs that we used in experiment 1 are connected to channel 5. And again, channel 18 is used to control our GPIO pins. Now we just need to connect these virtual pins to real pins in hardware.

Experiment 5: GPIO and software SPI

The MAX1000 board sports a type LIS3DH acceleration sensor made by ST. It would be desirable if the CPU could talk to this device, and for that purpose we will use the CPU's general-purpose I/O pins as mentioned above. These pins will be connected to the LIS3DH's interface pins. First we arrange for the physical pins to be configured as inputs or outputs in the top-level module of the Verilog code as follows.

```
...
input LISsenINT1,
input LISsenINT2,
input LISsenSDO,
output LISsenCS,
output LISsenSPC,
output LISsenSDI
...
```

Now we can connect up the LIS3DH pins to the GPIO signals on channel 18 of our DIY CPU inside the FPGA using a series of `assign` statements as follows.

```
assign GPIOinp[0] = LISsenINT1 ;
assign GPIOinp[1] = LISsenINT2 ;
assign GPIOinp[2] = LISsenSDO ;

assign LISsenCS = GPIOoutp[0] ;
assign LISsenSDI = GPIOoutp[1] ;
assign LISsenSPC = GPIOoutp[2] ;
```

These two modifications to the Verilog code are sufficient to enable the soft-core CPU to read and write data using INPA and OUTA instructions respectively. This illustrates how we can extend the peripherals already available in the CPU to create our own peripheral functions. We will now see how to write a C program that lets the CPU talk to the peripheral device.

Do it all in C

We have already seen how it is possible to include assembler code in our C programs, and this facility makes talking to the CPU's peripherals (and thence to external hardware) very easy. The example in **Listing 1** originates from the file *experiment5/IOlib5.c*.

To understand this code we need to know a bit about how the compiler works. When an arithmetic expression is evaluated the result is placed in R0. Also, R0 is used to hold the return value from a function call. Armed with this knowledge, and looking at the comments in the code, we can see how it all fits together. We are now in a position to write a function in C that can be used to access the LIS3DH acceleration sensor: we will write code to implement the SPI interface by 'bit banging' the GPIO pins. **Listing 2** shows some example C code to achieve this. So now to the experiment proper.

Action stations

The SCCCP development environment is launched by executing the file *c.bat* in *experiment5/experiment*. Then clicking on

Listing 1. Reading from and writing to a peripheral.

```
GPIOchannel       EQU  18

GPIOout(int p){
  p ;                 // 'calculate' the value of p and leave it in R0
#asm                  // now a bit of assembler code
  OUTA  GPIOchannel   // write value in R0 to GPIO channel
#endasm
}

GPIOin(){
#asm                   // switch to assembler code
  INPA  GPIOchannel    // read value from GPIO channel into R0
#endasm                // switch back to C
}
```

the 'MAKE' button will compile the C code.

```
c.bat:
Java
-jar c:\SCCCP\CompilerAsJar\sCCCP.jar
-s experiment5.c
-p ..\..\Processing\UploadCodeViaSerial1v01\
```

The effect of the *-p* option is that when the 'MAKE' button is clicked on, the resulting object code will be written to the upload directory with file name *mem1.java*.

The next step is to synthesize the Quartus project *experiment5.qpf* in the working directory, and load the resulting bitstream file into the FPGA. The uploader will now be sitting in the FPGA, ready to receive data and program memory contents over the service interface. To test this, use a terminal emulator program to send an 'x' character and verify that you receive a string of hexadecimal characters in response.
To upload the memory contents we launch the sketch *UploadCodeViaSerial1v01.pde*, which runs in the Processing programming environment. This will load the application program into the FPGA: the example here reads values from the three channels of the LIS3DH acceleration sensor and outputs them over the user serial port at 115200 baud. The results might appear as follows.

```
x= -00000032 y= -00000128 z= +00016576
x= -00000048 y= -00000224 z= +00016688
x= -00000064 y= -00000240 z= +00016544
x= -00000032 y= -00000224 z= +00016640
x= +00000016 y= -00000256 z= +00016704
```

If you now move and turn the MAX1000 board you should see the acceleration readings change.
Should you now wish to make changes to the programming running in our CPU, you simply need to compile the new version of the code and send it to the MAX1000 using the uploader sketch. This process is considerably quicker than resynthesizing the whole CPU from scratch every time. When your program is finally running as it should, the upload facility can be disabled in the CPU Verilog project by replacing the lines of *MainInclude.txt* that read

```
`define bootload
//`define standalone
```

by the following lines.

```
//`define bootload
`define standalone
```

The result of this change is that the data and program memory contents will be incorporated into the CPU design at synthesis time.

@ WWW.ELEKTOR.COM

→ MAX1000 FPGA development board
www.elektor.com/max1000

→ E-book: 'Microprocessor Design Using Verilog HDL'
www.elektor.com/verilog

Web Links

[1] The SCCC Project (1), ElektorLabs magazine 2/2019: www.elektormagazine.com/180394-01
[2] The SCCC Project (2), ElektorLabs magazine 3/2019: www.elektormagazine.com/180394-B-01
[3] Project page for this article: www.elektormagazine.com/180394-C-01
[4] Processing: http://www.processing.org

Listing 2. SPI communication using bit-banging.

```
int GPIOoutShadow ;
#define bitMaskLISsenINT1 1
#define bitMaskLISsenINT2 2
#define bitMaskLISsenSDO  4

#define bitMaskLISsenCS   1
#define bitMaskLISsenSDI  2
#define bitMaskLISsenSPC  4

LISinit(){
  GPIOoutShadow= bitMaskLISsenCS | bitMaskLISsenSPC | bitMaskLISsenSDI ;
  GPIOout(GPIOoutShadow) ;
 }

LSIclockPulse(){
  GPIOoutShadow = GPIOoutShadow & (~bitMaskLISsenSPC) ;
  GPIOout(GPIOoutShadow) ;
  GPIOoutShadow = GPIOoutShadow | bitMaskLISsenSPC ;
  GPIOout(GPIOoutShadow) ;
  }

SPItxrxByteMSBfirst(int vTX){
  int vRX ;
  int k ;
  int GPIOinp ;
  vRX=0 ;
  for( k=0 ; k<8 ; k++){
    if(vTX & 0x80) {
         GPIOoutShadow =GPIOoutShadow | bitMaskLISsenSDI ;
       }
        else {
          GPIOoutShadow =GPIOoutShadow &( ~ bitMaskLISsenSDI );
     }
    LSIclockPulse() ;
      vRX = vRX << 1 ;
      GPIOinp=GPIOin() ;
      if( GPIOinp & bitMaskLISsenSDO ) { vRX = vRX |1 ; }
      vTX=vTX<<1 ;
  }
  return vRX ;
}
```

Experiment 5 is one of our more complicated endeavours, as it involves a disparate range of files, programs and components. If something fails to work as it should, then the first things to check are the following: are all the paths specified in the batch files correctly set to reflect the installation? And are the COM port numbers and baud rates correctly set for the serial interfaces?

When you do have everything up and running you can then take a look at the code in *experiment5.c* in order to learn how it goes about outputting data using the user UART.

The code drives the acceleration sensor using a bit-banging technique. It would also be possible to implement the SPI functionality directly in the FPGA, but for our simple and non-time-critical application the bit-banging approach is much more straightforward.

In the next installment of this series we will take a look at how to generate analog signals, with the help of sigma-delta DACs and interrupts. Stay tuned!

(180394-C-02)

Student Demonstration Time!



Photo credit: Johan van Uden.

TUE/InMotion: Revolutionary Battery Pack Design

Kjeld Teunissen (The Netherlands)

The InMotion student team from the Technical University Eindhoven (TUE) in The Netherlands is grabbing this space to publicize they're working on 'Electric Refuelling', a concept which aims to offer a solution to recharge electric cars as fast as you would refuel those with a combustion. Interesting, right? Let's see what they're up to...

The InMotion student team, which I am a member of, consists of more than 50 young and ambitious students from the Eindhoven University of Technology and the Fontys University of Applied Sciences. Student teams offer a unique collaboration between universities and the industry, permitting work at the very edge of technology. We are working on a concept called 'Electric Refuelling', which is the ability to charge your car in the same amount of time as you would refuel a car with a combustion engine. We intend to showcase the potential of electric refuelling by joining the **Garage 56 Class** during the **24 Hours of Le Mans** race with a fully electric endurance race car in 2023. For this, we need to develop our own custom-made battery pack and this is my responsibility for this year. So, in case you have ever wondered

Is your team working on something cool?

Are you on a student team with a cool or amazing achievement to report on in the field of electronics?
Then email Editor@elektor.com, subject “Student Demonstration Time”.



- how electric vehicles are powered
- how it’s possible to power a vehicle with cells so small they fit in the palm of your hand

then this article is just for you.

First things first

If ever you wanted to design a battery pack no matter its application, you should always clearly define what you want in a set of requirements. The four main requirements are the (minimum, nominal and maximum) voltage, capacity, weight, and power. Furthermore, you should always take safety features and regulations into account.
Current battery technology seems to force us into a dilemma. For one, we can aim for a high-capacity battery pack that allows you to take more energy with you and therefore cover longer distances. Alternatively, we can build a high-power battery pack that’s more suitable to high-power applications and faster charging. Since the focus of InMotion is on developing fast-charging technology, our battery pack of choice is the latter. So, we are now developing a battery pack with a lower capacity than offered by some commercial vehicles. Still, this allows us to develop technology that can be used when battery technology has improved to a level where high-power battery packs provide enough capacity for commercial applications.



The InMotion team (photo credit: Bram Naus).

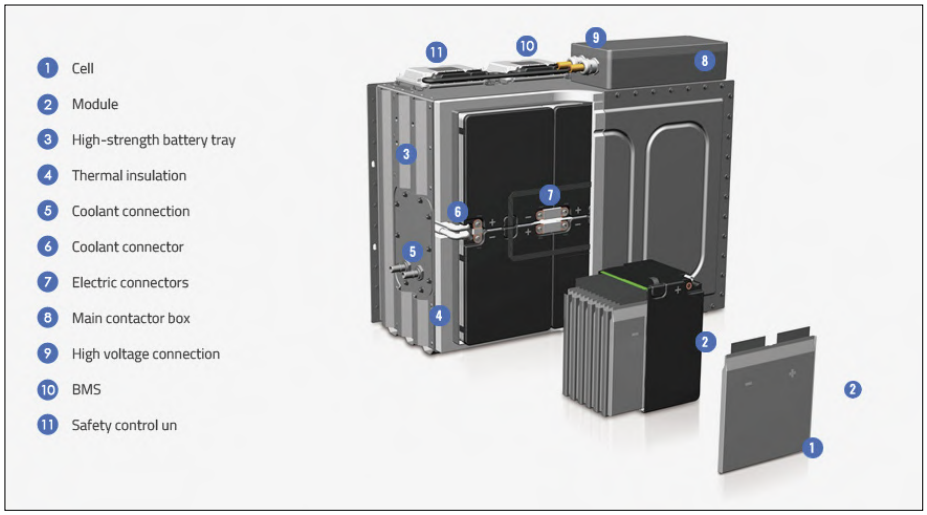


Figure 1. An example of a battery cooling pack made by Kokam [1].

Structure

As mentioned before, a battery pack is made up of lots of smaller battery cells. Several cells are bundled in a module, and these modules are then configured in a battery pack (see also **Figure 1**).

Cell

The cell type is probably the most important decision in the process. It is the defining factor in the battery pack's performance. InMotion uses pouch cells because they're better suited for fast charging than cylindrical cells.

They can take more abuse, usually have a lower internal resistance which in turn generates less heat, and they have a larger surface area which makes it easier to cool the cells.

Module

The design variables of a single module are determined by the free space required around each cell. With pouch cells the space above, below and on the sides of the cells are all parameters that need to be determined or acquired. These usually depend on the placement of the tabs and the type of cooling used.

Pack

With the dimensions of the modules determined, they're ready for stacking to make a pack. This will determine where cooling lines need to be run, as well as how the modules need to be connected electrically inside the pack.

Fancy software

At InMotion the development stage is very dynamic. It allows us to change things on-the-fly. It also means that not all our requirements are set in stone, and we're trying to find a set that fits our goals. The battery pack is a good

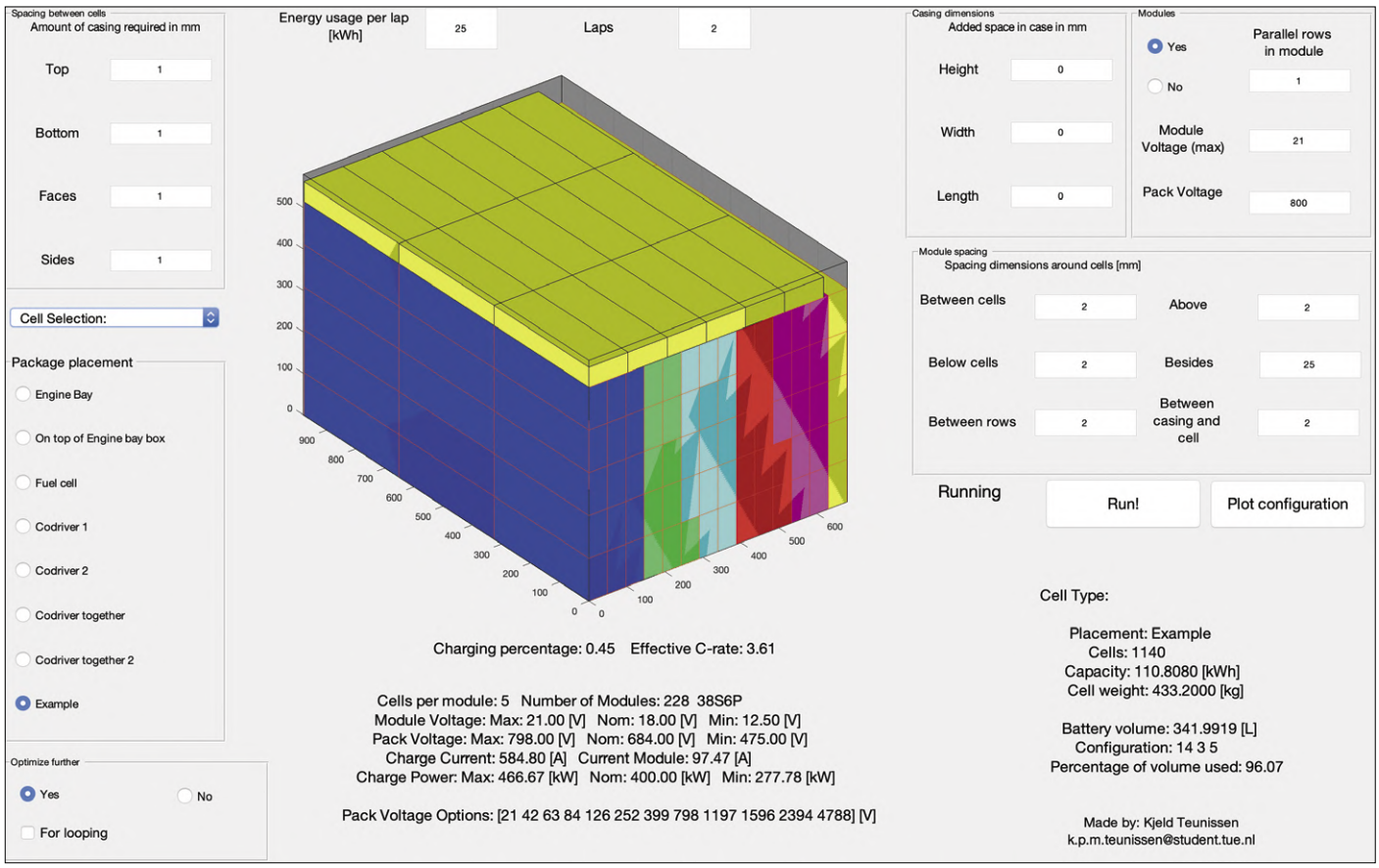


Figure 2. A specially written program helps to design the battery pack.

example of this as it can be configured in literally a million ways. But how to find out which configuration works best? To avoid having to generate a model for every single configuration, I have made a battery pack configurator that optimizes the pack for different input parameters. To ensure ease-of-use and to allow the user to try out different configurations without having to understand the underlying code, I created a GUI (see **Figure 2**). It basically takes all the design variables as input and then tries to fit as many modules in a given space, stacking them in the same direction. In the space that's left, it will once again try to fit as many modules as possible, even if that means the orientation of some modules is different than that of others. The script uses a heuristic method rather than an analytic approach to this space optimization problem. This is done because it is easier to write and still reaches local minima. It is also extremely efficient and fast. On an average PC the script creates 190 optimal configurations, selects the best for a given set of preferences, and plots it within 5 seconds. Within this time it also saves all created optimal configurations, allowing them to be filtered for different preferences. If a different configuration turns out to be better, it's easily plotted using the GUI.

The GUI calculates the expected performance of the battery pack. It tells you:

- the expected capacity;
- the number of modules connected in series and in parallel;
- the expected cells weight;
- the voltage difference between a full pack and an empty one.

Additionally it calculates the charging currents while indicating the heat generation as well as the charging power range that will be necessary to charge the pack in 7.5 minutes.

Achievements

With Electric Refueling we aim to be able to charge an electric car in just a few minutes. If successful, it will undoubtedly accelerate the energy transition from petrol to electric in the automotive industry. To show the world how fast electric racing can be, we have created the "Fusion", an electric formula car that currently holds the electric lap records at Circuit Zandvoort, TT Circuit Assen (both in The Netherlands) and Circuit Zolder (Belgium). Our concept car "Vision" is the true embodiment of future electric mobility. It is designed to be 10 times more aerodynamically efficient than a Formula 1 car. It is truly the endurance racer of our dreams. ⏮

180732

Web links

[1] Website for Kokam: http://kokam.com/modulepack

[2] InMotion website: www.inmotion.tue.nl/

Advertisement



ESP32 Battery Monitor

Measures three channels via Wi-Fi and Thingspeak



By **Laurent Labbe** (Hong Kong) and **Luc Lemmens** (Elektor Labs)

This project shows the simplicity and ease of measuring analogue values with an ESP32 board and to send the values via Wi-Fi to a router, and from there, to the ThingSpeak website where the data are saved and displayed.

At the moment, Laurent is working on several projects powered with solar energy. He was looking, amongst other things, to develop a solar power source for his Nixie watch [1]. By its nature, solar radiation is not dependable, so it may take some hours or even days to test a power supply model. He developed three different modules, each charging three identical small batteries (130 mAh) with a resistor to simulate the power drawn by the watch. Then he read the battery voltage on the three modules and sent them to the free ThingSpeak platform [2] for the Internet of Things (IoT).

Measurements

The measurements are taken by an ESP32-PICO-KIT-V4 development board powered by USB (5 V). As the device is used outside, with no mains power available, the author used two external batteries (powerbanks), one small (1000 mAh) and one large (10,000 mAh). The smaller is connected permanently to power the ESP32 board. It is recharged by the larger one which is replaced almost every day so that the system is always working. If only a single battery was used, measurements would be lost during the recharge of the battery powering the ESP32.

Three I/O pins of the ESP32 (**Figure 1**) are used to take the measurement sof the three channels/batteries. Each pin has its own voltage divider and a 100 nF filter capacitor. In the sketch, this attenuation is represented by the variable `Coef_Vx` and its value may be adjusted to compensate for tolerances in the external voltage divider.

The analogue/digital converted of the ESP32 is equipped with an internal attenuator which may be set by the Arduino sketch with the function `analogSetAttenuation` to one of the following values: 0 dB, −2.5 dB, −6 dB or −11 dB; corresponding to factors of 1 ; 0.75 ; 0.5 or 0.28.
By default, the full scale voltage of the ADC is 1.1 V. This voltage is the maximum that can be read (as a function of the number of bits configured for ADC1: 4095 for 12 bits, 2047 for 11 bits, 1023 for 10 bits, 511 for 9 bits). To read higher voltages (up to the maximum voltage for the pin, generally 3.3 V = Vdd), an attenuation greater than 0 dB must be set for the ADC channel in use. **Table 1** sets out the the settings and the ranges which can be used. For an attenuation of −11 dB, the

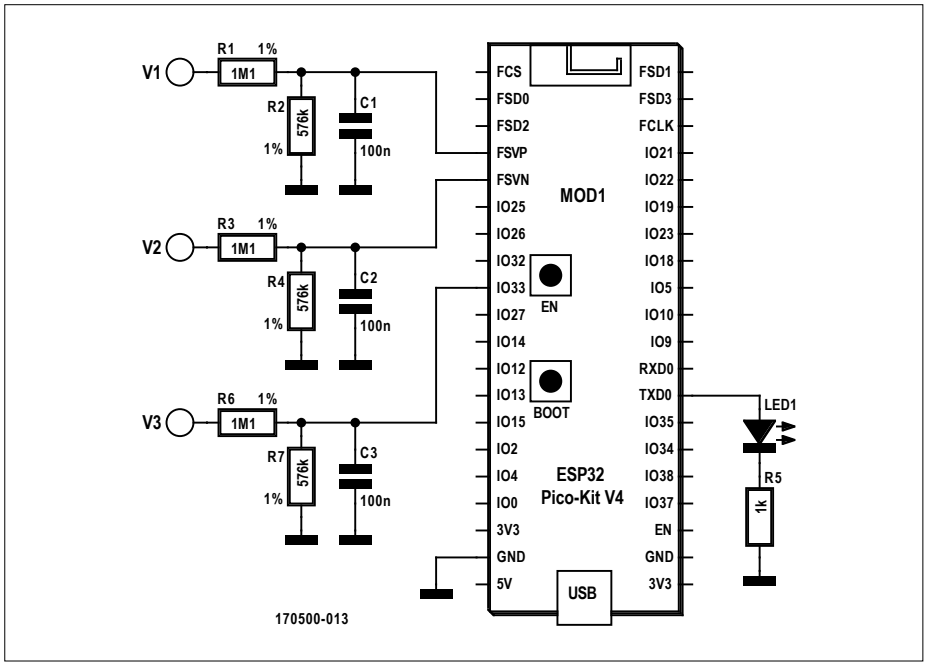


Figure 1. Circuit diagram: the ESP32 board takes care of most of the work.



maximum voltage is limited by VDD_A, and not by the full scale voltage.
The circuit was designed to monitor LiPo batteries at 4.2 V and the analogue inputs are calculated to support a maximum input voltage of 5 V. with a voltage divider using resistors of 1.1 MΩ and 576 kΩ, the inputs of the ESP32 board are specified for a voltage of ±1720 mV for a voltage input of 5 V, which is the maximum input voltage recommended for and attenuation of −6 dB.Of course, you can modify the input circuit if you need a different input voltage range, and/or change the internal attenuation of the ADC of the ESP32 board.

Note: you can only apply DC voltages to the inputs which are within the range defined by the settings of the ADC. In addition, there is no protection against reverse polarity or overvoltages!
Pin GPIO17 of the ESP32 board drives a LED which lights during voltage measurements and data transmission. Then the LED goes off and the ESP32 board goes into sleep mode until it automatically wakes up for the next measurements. This minimises energy consumption from the external batteries.
You can easily wire up the circuit on a breadboard or a piece of stripboard.

Create a ThingSpeak account

Laurent uses the online platform ThingSpeak for a weather station project. So he naturally opened a page there to store the measurements taken for his watch batteries (see [3])..
ThingSpeak is a free web service with open source code, which allows you to gather and store in the cloud data from sensors, and develop applications for the Internet of Things. Sensor data can be sent to ThingSpeak from Arduino, Raspberry Pi, BeagleBone Black and similar boards.
Also, it is possible to use functions from various MATLAB (from MathWorks) toolboxes on a ThingSpeak connection. For that it is necessary to be connected with a MathWorks account with an active licence in order to use these toolboxes. If you're starting this project and you don't yet have a ThingSpeak or MathWorks account, Go to www.thingspeak.com and click `Get Started For Free`, then create an account (**Figure 2**). Then click on `Connexion` to access your account and fill in your ID information. Then create an account where the Battery Monitor data will be collected (`New Channel`), give your channel a logical name which will be easily recognised as your battery monitor data. In the `Channel Settings`, add three fields to sort the data which the Arduino sketch will send in the form of strings and label them (in our case V1, V2 and V3 as in **Figure 3**). Now choose the `API keys` tab, here you will find the *Write API key* which you must copy into the sketch of the ESP32 board to identify the ThingSpeak channel(**Figure 4**).

Battery Monitor Sketch

The `Battery_monitor_elektor_3.ino` sketch written in C is not very complicated, but before compiling it and uploading it to the ESP32 board, you need to

Table 1. Theoretical full scale of the ADC and actual full scale available, as a function of attenuation settings if VDD_A = 3.3 V

ADC Attenuation	Theoretical full scale	Actual range for the most accurate results
0 dB (ADC_ATTEN_DB_0)	1.1 V	100 - 950 mV
2.5 dB (ADC_ATTEN_DB_2_5)	1.5 V	100 - 1250 mV
6 dB (ADC_ATTEN_DB_6)	2.2 V	150 - 1750 mV
11 dB (ADC_ATTEN_DB_11)	3.9 V	150 - 2450 mV

(Source : https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/adc.html#adc-calibration)

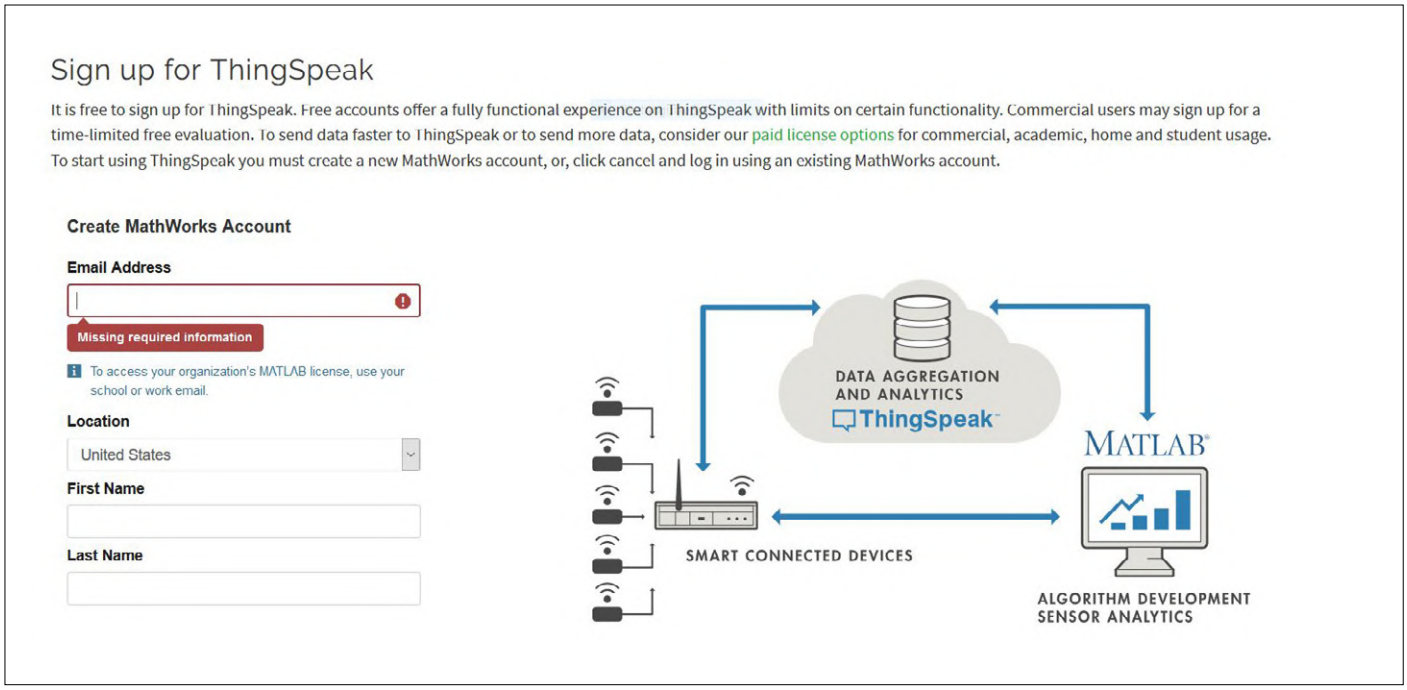


Figure 2. Creation of a ThingSpeak account.

modify it so that the board connects to you Wi-Fi network and sends the data to your ThingSpeak channel.
On line 68 of the sketch, enter the *Write API key* that you got from the ThingSpeak site to identify your channel. Then enter the SSID and the password for your local area network in lines 70 and 71 respectively. Then upload the sketch to the ESP32 board and open the serial monitor of the Arduino IDE in the `Tools` menu. Make sure that the transmission speed of the monitor is set to 115200 and you should receive a message from the ESP32: `Battery monitor V1.0 Start` followed by the first three voltage measurements.
After each measurement cycle the ESP32 passes into sleep mode. `Boot number: x` indicates how many cycles have run since the beginning of the session (i.e. how many measurements have been made until now).
Then the ESP32 board (re)connects to you Wi-Fi network and shows that the connection is successful with the IP address of the ESP32, or sends the message `WiFi cannot connect` if the attempt fails.
You can set the measurement interval by modifying `TIME_TO_SLEEP` on line 45. Note that only the `setup` function is executed, it performs all the process-

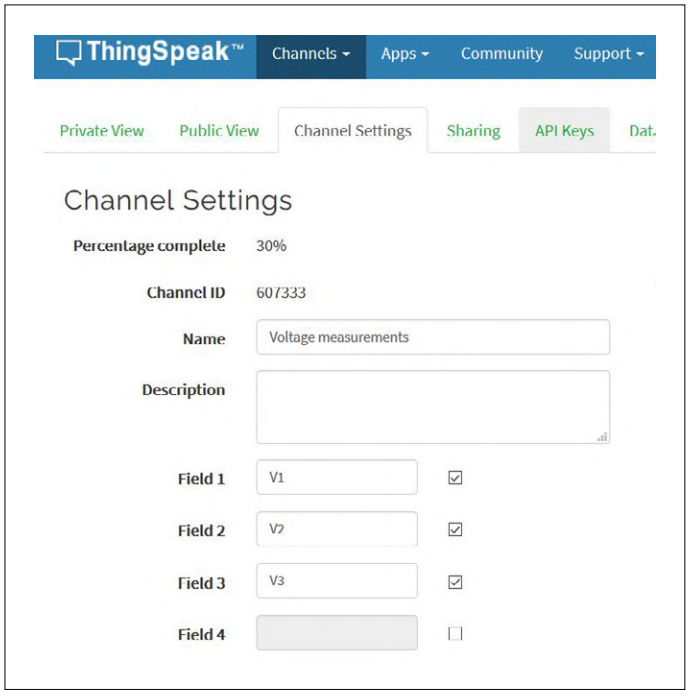


Figure 3. Channel settings on ThingSpeak.

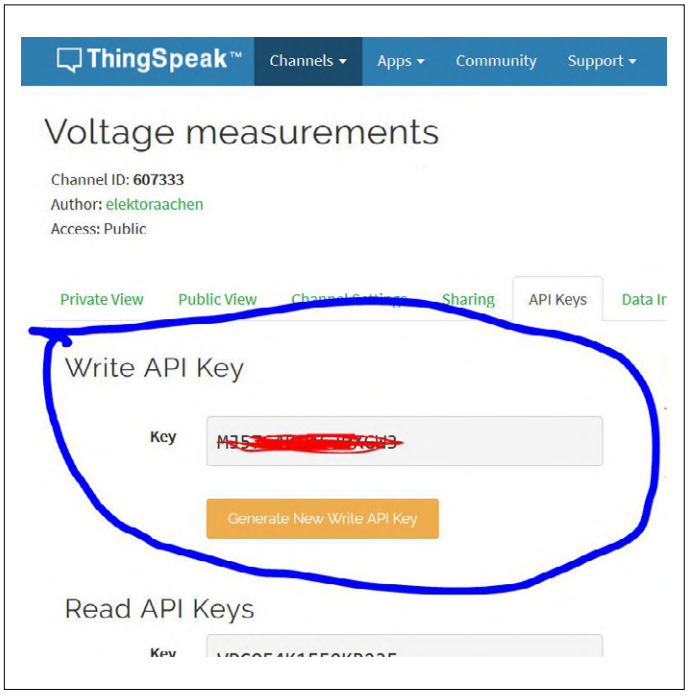


Figure 4. API key to recopy into the sketch.

ing (measurements and data transmissions) and puts the ESP32 into sleep mode until an internal timer wakes it up for the next cycle of measurements. The `setup` function is then executed again. The `loop` function is empty, and it is never executed.

If this is your first project with the ESP32 board in the Arduino IDE, don't forget to install the ESP32 boards in this environment. You will find numerous tutorials on the Internet (See for example [4]). Note: select the right COM port for the connection to the ESP32-PICO-KIT-V4 in the menu `Tools -> Port` and choose `ESP32 Pico Kit` in `Tools -> Board`.

This project is simple and quick to get working as there is no printed circuit board to get and wire up. It opens the door to all sorts of other measurements and lets you get a foot in the cloud.

170500-02



Web links

[1] Elektor Labs Nixie watch page: www.elektormagazine.com/labs/nixie-watch-1

[2] ThingSpeak Platform: www.thingspeak.com

[3] ThingSpeak battery monitor public page: http://thingspeak.com/channels/329068

[4] Installation of ESP32 boards in Arduino IDE: http://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions

[5] Elektor Labs Battery monitor page: www.elektormagazine.com/labs/simple-wifi-multimeter

[6] Article support page: www.elektormagazine.com/170500-02

Advertisement



Differential Probe Power Supply

An ultra-simple power solution using USB

By **Dr. Martin Oppermann**

In our July & August 2016 issue [1], Elektor author Alfred Rosenkränzer presented a wonderfully simple differential probe for oscilloscopes, based around the AD8479 precision differential amplifier. I built one of them immediately and found it particularly suitable for measurements in the audio range, for example. But one small detail was not covered in this article: the power supply used for the probe (or for several of them). Practically all modern oscilloscopes have a USB port on the front panel for data output. Since the power consumption of a probe is very low, it makes sense to use this socket to power the probe. **Figure 1** shows the schematic of the circuit I decided upon.

A standard USB cable is used to connect the probe power supply to the USB outlet on the oscilloscope; the type is unimportant, as even a charging cable is adequate. C1 and C2 filter the DC voltage from the USB port and LED1 indicates that the power is on. The heart of the circuit is the DC-DC converter IH0512S from XP Power [2]. It generates symmetrical ±12 V from +5 V with a maximum output current of ±84 mA, and it is galvanically isolated (with an electrical barrier between the input and output to improve safety)! The isolation voltage is 1,000 V, meaning there should be no ground problems when making measurements. The output voltage is filtered using various capacitors and ferrites, which can be omitted when working at the AD8479's maximum measurement frequency of 130 kHz or replaced by 0-Ω resistors or wire straps.

To connect Rosenkränzer-type probes, I used 3.5-mm stereo jack sockets. These connectors are nice and compact, although they do have the disadvantage that their contacts are short-circuited momentarily when the jack plugs are inserted. In order to prevent damage, 100-Ω resistors are provided in each output circuit. Since the total current consumption of a probe is in the range of a few milliamperes (the AD8479 datasheet indicates about 850 µA; the only additional current drawn is by the operating voltage indicator LEDs D5 and D6), the voltage drop arising across these resistors is not a problem.

The circuit was built on a small piece of double-sided board, mainly with SMDs, with the underside used only as a ground plane. The smallest SMD package used is 1206, which is still perfectly easy to solder by hand. **Figure 2** shows the layout that I created in Autodesk Eagle, which can be downloaded from the project page [3].

The PCB fits inside a small, translucent blue plastic enclosure from Hammond Manufacturing (type 1551RTBU).

Lastly a tip. In my own setup I used 78L09 and 79L09 (i.e. ±9 V) voltage regulators for IC2 and IC3 of the probes, to increase their measuring range to ±8.7 V.

(180177-02)

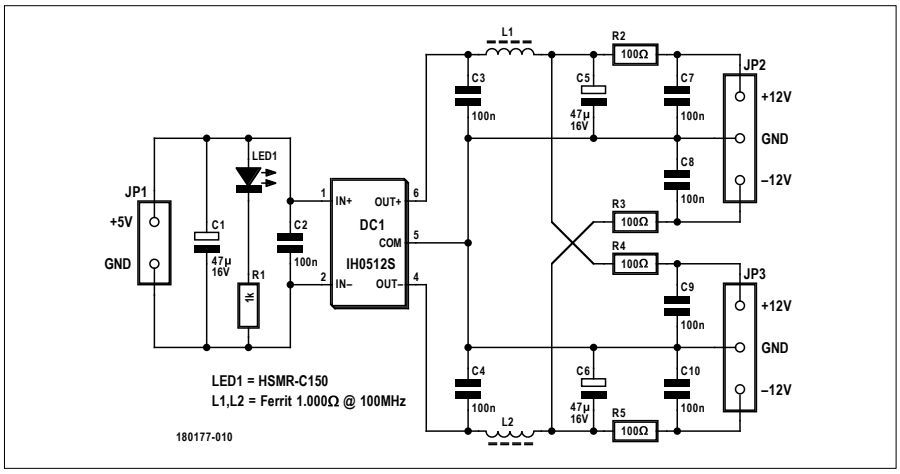


Figure 1. Schematic of the power supply circuit, equipped for two probes.

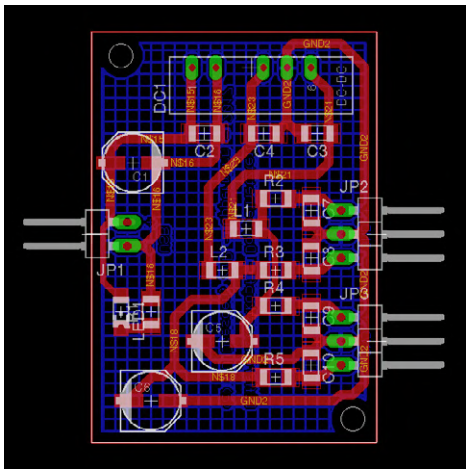


Figure 2. PCB layout and component placement.

Web Links

[1] Rosenkränzer, Alfred: A Simple Current Sensor Probe, Elektor July/August 2016: www.elektormagazine.com/150182

[2] 2W DC-DC Converter IH0512S: www.xppower.com/Portals/0/pdfs/SF_IH.pdf

[3] Project page: www.elektormagazine.com/180177-02

Display Serial Data on a Web Page

using PHP or Python scripts



By Clemens Valens (Elektor Labs)

Occasionally it may be useful to visualize a continuous stream of data arriving through a serial connection as a web page in a browser. But how do you go about? May we suggest a bit of scripting?

Basically, the problem boils down to periodically creating a webpage containing the latest serial data. A web browser then can display the page, be it on the same computer or over a network. Therefore, what is needed is a small program that continuously converts the serial data into for instance HTML or PHP files. Aha! Isn't PHP a programming language for Internet related applications? Yes, it is. So, can we use PHP to get the job done? Yes, we can. But there are other ways too, as we will see.

Use automatic page refresh

It all starts by remembering that HTML allows the use of a meta tag that instructs the browser to reload a page periodically:

```
<meta http-equiv="refresh" content="10">
```

This tag tells the browser to reload the page containing the tag every ten seconds. (In case your browser does not support this tag, it can be replaced by a piece of JavaScript. See download at [1]) If we create a web page that includes this tag and point the browser to it, the latter will keep reloading it every ten seconds (the timeout can, of course, be another value). If we rewrite the page every ten seconds and put fresh serial data into it, then the browser will show it too.

Split the task in two

It is also possible to put the refresh tag in a PHP file instead of an HTML file and the browser will do the same thing. The PHP file then could also contain a script to read the data from the serial port. This is the point where things become complicated because PHP doesn't natively support serial ports. And even if it did, it would mean that every time the browser requests the (latest version of the) page, the script would have to open the serial port, get some data and close the port again. Data received outside this window is lost. Furthermore, some Arduino-like systems may reset when the serial port is opened, making the setup useless. A solution to this is to split the process in two subprocesses:

- Process 1: A script to continuously read the serial port and update a data file that is imported by the PHP webpage (**Listing 1**);

Listing 1: A PHP script that reads data from the serial port and writes it to a file named “data.txt”.

```
<?php

// Linux $comPort = “/dev/ttyACM0”;
$comPort = “COM15”;

include “php_serial.class2.php”;
$serial = new phpSerial;
$serial->deviceSet($comPort);

// On Windows (10 only?) all mode settings must be done in one go.
$cmd = “mode “ . $comPort . “ baud=115200 parity=n data=8 stop=1 to=off xon=off”;
$serial->_exec($cmd);
$serial->deviceOpen();

echo “Waiting for data...\n”;
sleep(2); // Wait for Arduino to finish booting.
$serial->serialflush();

while(1)
{
  $read = $serial->readPort();

  if (strlen($read)!=0)
  {
    $fp = fopen(“data.txt”,”w”);
    if ($fp!=false)
    {
      fwrite($fp,trim($read));
      fclose($fp);
    }
  }
}

?>
```

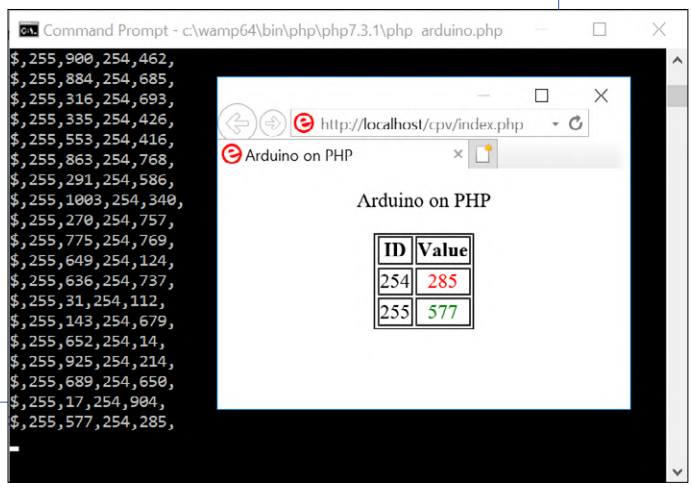


Figure 1: The dynamic PHP file generated by the PHP script is served by a WAMP web server to our browser (see its address bar). It presents the comma-separated values read from the serial port data in a basic table. Values below 500 are printed in red, the others are green. The title of the Command Prompt window shows the command to execute the script.

- Process 2: A browser that periodically reloads the PHP webpage so that it can refresh the data (**Listing 2, Figure 1**).

Oops, web server needed...

Doing this solves the serial port opening and closing problem and the data loss resulting from it, but it requires a script to run in the background. If this is a PHP script, then the computer must be able to execute PHP scripts. Also, you need a web server to feed the PHP webpage to a browser. If not, the browser will simply show the PHP code making up the page instead of the page itself. The traditional way of getting this done is by installing a so-called “AMP” or “WAMP” package. AMP stands for Apache-MySQL-PHP, the ‘W’ is for Windows, and together they form an all-singing, all-dancing web server.

Don’t use PHP...

We have tried this method and managed to get it to work, but not without problems. Besides difficulties in setting up the web server, the main problem we encountered was getting PHP to reliably open a serial port to receive the data. Searching the Internet, there seems to be only one PHP library for serial communications, PHP Serial. All others appear to have been derived from it. As the author mentions on the GitHub page [2] “*Windows: it seems to be working for some people, not working for some others.*” We clearly were in the second group. To get serial communications to work with PHP, we first had to open and immediately close the port with a serial terminal program (Tera Term for instance); impossible to get it going otherwise. We therefore ditched the PHP method and went for Python instead.

... use Python instead

Python 3 with pySerial [3] turned out to work perfectly fine on our Windows 10 test computer so we wrote a script to read data from the serial port and write the webpage containing the data. Now that there is no longer a need for PHP, the Python script might as well produce a plain HTML file (**Listing 3, Figure 2**). All the data formatting done by the PHP web page can be done directly in Python. Everything is now in one place. The web page may still be served up by a web server (pun intended), but a browser can also display and refresh the page without one. Therefore, there is no longer a need for a (W)AMP package either, making everything much simpler.

Listing 2: This PHP webpage formats the contents of a file named “data.txt” as a table.

```
<?php

$page_title = “Arduino on PHP”;

// Start of HTML page.
echo “<!DOCTYPE HTML PUBLIC ‘-//W3C//DTD HTML 4.01//EN’ ‘http://www.w3.org/TR/html4/strict.dtd’>”;
echo “<html>”; // Page begin.
echo “<head><title>”,$page_title,”</title>”; // Head begin.
echo “<meta http-equiv=’refresh’ content=’1’>”;
echo “<meta http-equiv=’Content-Type’ content=’text/html; charset=UTF-8’>”;
echo “<link rel=’shortcut icon’ href=’favicon.ico’ />”;
echo “<link rel=’icon’ type=’image/x-icon’ href=’favicon.ico’ />”;
echo “<link rel=’icon’ type=’image/png’ href=’favicon.png’ />”;
echo “</head>”; // Head end.
echo “<body><center>”; // Body begin.

echo “<p>”,$page_title,”</p>”; // Page title.

// Create a table from data file.
$handle = fopen(“data.txt”,”r”);
if ($handle!=NULL)
{
  // Read one line from the file, then close it.
  $data = fgets($handle);
  fclose($handle);

  // Synchronise to the data.
  if ($data[0]==’$’)
  {
    // Remove whitespace.
    str_replace(‘ ‘,’’,$data);
    // Split data in fields separated by ‘,’.
    // Expected format: “$,id1,value1,id2,value2,CRLF”
    list($startchar,$id1,$value1,$id2,$value2,$newline) = explode(“,”,$data);
    // Create array from list.
    $numbers = array($id1=>$value1,$id2=>$value2);
    // Sort array in ascending key order.
    ksort($numbers);

    // Table begin.
    echo “<table border=’1’ border-spacing=’5’ style=’text-align:center;’>”;
    echo “<tr><th>ID</th><th>Value</th></tr>”;
    foreach ($numbers as $x => $x_value)
    {
      echo “<tr>”; // Table row begin.
      echo “<td>”, $x, “</td>”; // Table column 1.
      echo “<td>”; // Table column 2 begin.
      if ($x_value>=500) echo “<font color=’green’>”;
      else echo “<font color=’red’>”;
      echo $x_value;
      echo “</font></td>”; // Table column 2 end.
      echo “</tr>”; // Table row end.
    }
    // Table end.
    echo “</table>”;
  }
}
echo “</body>”; // Body end.
echo “</html>”; // Page end.
?>
```

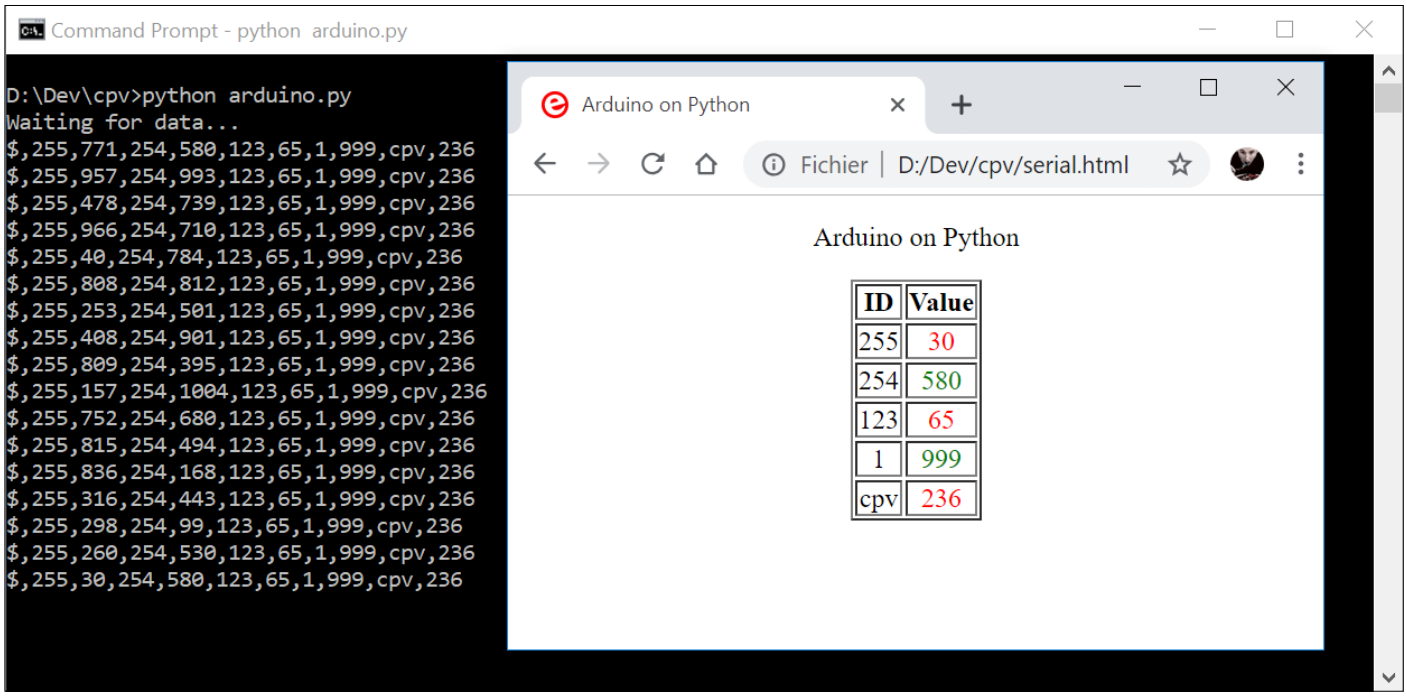


Figure 2: This time a Python script produces a dynamic HTML file (as can be seen from the browser's address bar).

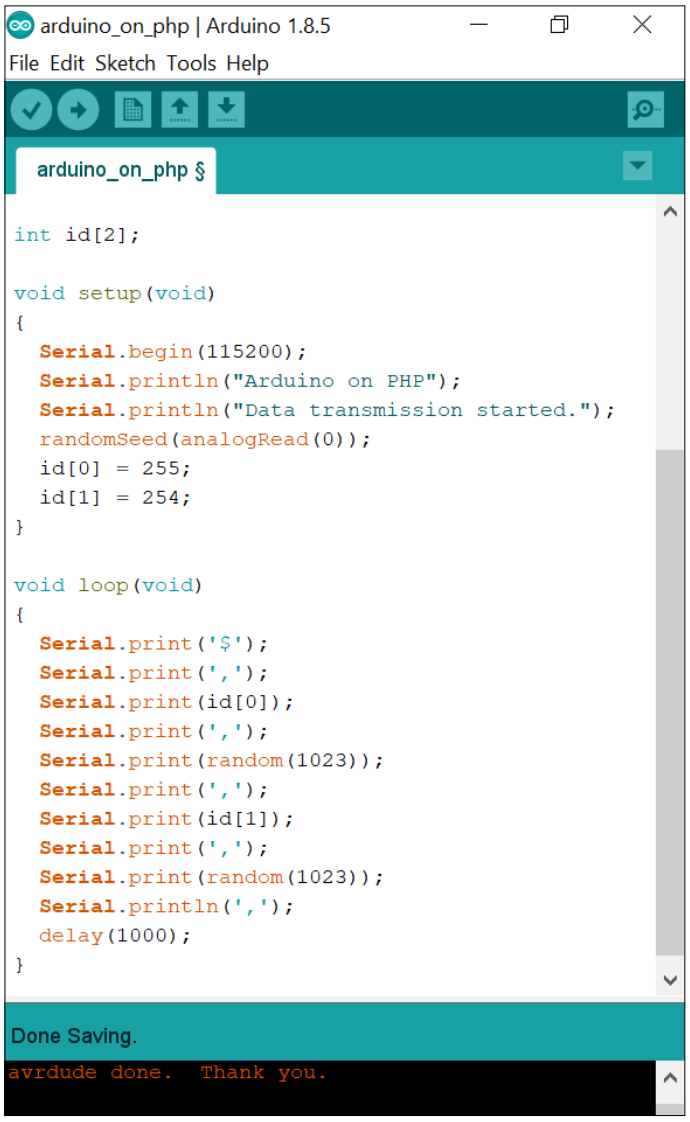


Figure 3: This Arduino sketch generates a serial data stream that can be used for script development, debugging, and testing.

Sounding off

In this article we presented a method of displaying serial data in a web browser. The method is neither new, exclusive, or "the best". If you know of another way — simpler, more elegant, whatever — please share it with us [1]. And, of course, you don't need Python to write the script, feel free to use any programming language capable of doing serial communications and writing files. The advantage of using Python with pySerial is that it will run on Windows, macOS, and Linux machines (and more).

The code developed for this article in the shape of PHP and Python scripts, and an Arduino sketch can be downloaded from [1]. ⏮

(170111-01)

Web Links

[1] Downloads for this article: www.elektormagazine.com/170111-01

[2] PHP Serial: https://github.com/Xowap/PHP-Serial

[3] pySerial: https://pypi.org/project/pyserial/

Listing 3: A Python script that reads data from the serial port and generates an HTML file for it.

```
import serial
import time

file_name = “serial.html” # Once created, open this file in a browser.

# Adapt serial port nr. & baud rate to your system.
serial_port = ‘COM15’
baudrate = 115200

page_title = “Arduino on Python”;

def write_page(data_list):
    fo = open(file_name,”w+”)
    # Start of HTML page.
    fo.write(“<!DOCTYPE HTML PUBLIC ‘-//W3C//DTD HTML 4.01//EN’ ‘http://www.w3.org/TR/html4/strict.dtd’>”)
    fo.write(“<html><head><title>”+page_title+”</title>”) # Page & Head begin.
    fo.write(“<meta http-equiv=’refresh’ content=’1’>”)
    fo.write(“<meta http-equiv=’Content-Type’ content=’text/html; charset=UTF-8’>”)
    fo.write(“<link rel=’shortcut icon’ href=’favicon.ico’ />”)
    fo.write(“<link rel=’icon’ type=’image/x-icon’ href=’favicon.ico’ />”)
    fo.write(“<link rel=’icon’ type=’image/png’ href=’favicon.png’ />”)
    fo.write(“</head><body><center><p>”+page_title+”</p>”) # Head end, body begin.

    # Table begin.
    fo.write(“<table border=’1’ border-spacing=’5’ style=’text-align:center;’>”)
    fo.write(“<tr><th>ID</th><th>Value</th></tr>”)
    for i in range(0,len(data_list),2):
        fo.write(“<tr>”) # Table row begin.
        fo.write(“<td>”+data_list[i]+”</td>”) # Table column 1.
        fo.write(“<td>”) # Table column 2 begin.
        fo.write(“<font color=’”)
        # Values >= 500 will be printed in green, smaller values will be red.
        if (int(data_list[i+1])>=500): fo.write(“green”)
        else: fo.write(“red”)
        fo.write(“’>”)
        fo.write(data_list[i+1])
        fo.write(“</font></td>”) # Table column 2 end.
        fo.write(“</tr>”) # Table row end.
    fo.write(“</table>”) # Table end.
    fo.write(“</body>”) # Body end.
    fo.write(“</html>”) # Page end.
    # Done, close file.
    fo.close()

s = serial.Serial(serial_port,baudrate) # Open serial port.
s.dtr = 0 # Reset Arduino.
s.dtr = 1
print(“Waiting for data...”);
time.sleep(2) # Wait for Arduino to finish booting.
s.reset_input_buffer() # Delete any stale data.

while 1:
    data_str = s.readline().decode() # Read data & convert bytes to string type.
    # Clean up input data.
    # Expected format: “$,id1,value1,id2,value2,...,CRLF”
    data_str = data_str.replace(‘ ‘,’’) # Remove whitespace.
    data_str = data_str.replace(‘\r’,’’) # Remove return.
    data_str = data_str.replace(‘\n’,’’) # Remove new line.
    data_str += ‘123,65,1,999,cpv,236’ # Add some more data
    print(data_str)
    # Split data in fields separated by ‘,’.
    data_list = data_str.split(“,”)
    del data_list[0] # Remove ‘$’
    # Write HTML page.
    write_page(data_list)
```

Bright Flashing LED

traditional analogue...

By **Bram Melse, MSc.** (The Netherlands)

A flashing LED? What is so special about that, you may well ask yourself. A microcontroller plus a handful of discrete parts, and Bob's your uncle – isn't it? In harmony with the emphasis on analogue electronics in this edition we present here a variant that is built entirely from discrete parts and hopefully triggers your own experimentation!

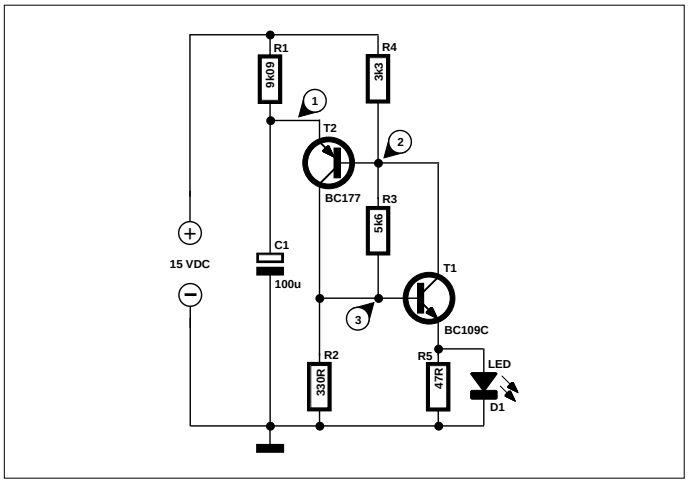


Figure 1: The schematic counts only a few parts. The transistors are not critical; just about any TUP and TUN should suffice.

The author has kept himself busy professionally with the design of (mainly) analogue chips. But to prove that you don't always have to use an IC, and also because it is sometimes just fun to figure out an analogue circuit, he presents here a flashing LED that needs no more than a couple of transistors and a handful of small parts. The result is drawn in **Figure 1**.

The operation of the circuit is best described as follows. For convenience we start with the situation where the voltage at node 1 is relatively low. A current flows towards node 3 via R3 and R4 (and therefore also R2), but the current is not large enough to turn T1 on.

Capacitor C1 is continually being charged through resistor R1; once R1 (via T2) also starts to supply current towards node 3, the sum of the currents becomes large enough for T1 to turn on. The voltage at node 2 will be pulled low and the circuit locks itself in this state (functions as a latch).

The current towards node 3, via resistor R3, now stops, but instead capacitor C1 is discharged (via T2) and the LED lights up brightly. As C1 discharges, the current through T2 towards node 3 diminishes until it is no more than what is supplied via R1, and that is not enough to keep T1 turned on. Node 2 will now be pulled high by R4, which turns T2 off. At this instant the entire cycle begins again.

Resistor R5 has an important function as a bypass: without this resistor, the LED would cause a threshold voltage that is too high for T1 to turn on, with the consequence that the latching action doesn't happen. The bypass current amounts to about 10% of the LED current.

Note that the LED flashes very brightly (with the component values indicated, at a rate of about 0.8 Hz); a simulation (in LTspice XVII) indicated a value of about 600 mA. Of course it is not ideal when the capacitor is discharged only through the transistors and the LED; you can experiment here with a current-limiting resistor (which, by the way, will also increase the duration of the flash).

Audio variant

If you have an appetite for that, you can also build an audio variant of this circuit. The circuit remains practically the same; only the component values need to be changed as follows:

- Resistors: R1 = 3k9; R2 = 180 Ω; R3 = 2k7; R4 = 1k8; R5 = 15 Ω.
- Capacitor: C1 = 0.2 µF.
- Transistors: T1 = BC109C; T2 = BC240B (not critical, any TUP and TUN will do).
- The loudspeaker (8 Ω) is connected between C1 and ground; the LED is omitted.

This variant generates a (somewhat objectionable) sound with a frequency of about 1 kHz. However that would be perfectly suitable for alarm applications.

The construction of the circuit is not very critical; a small piece of prototyping board is more than sufficient. ⏮

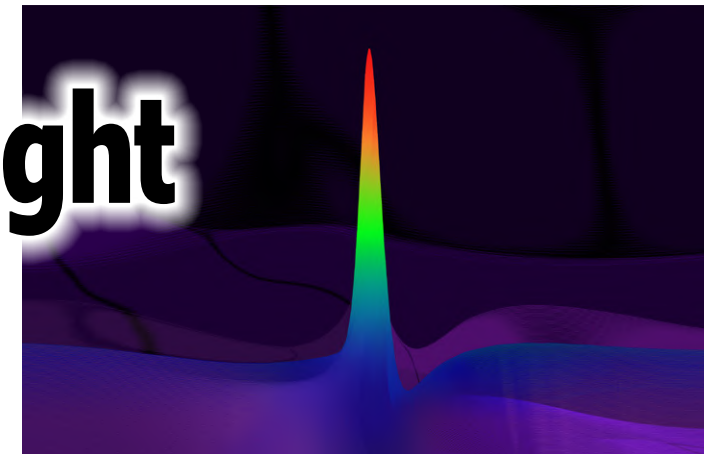
(180542-02)

@ WWW.ELEKTOR.COM

→ Book, Electronic Circuits for All
www.elektor.com/electronic-circuits-for-all

→ Book, The Art of Electronics, part 1 (analoue electronics)
www.elektor.com/the-art-of-electronics-3rd-edition

Heading Towards a Tsunami of Light



Researchers at Chalmers University of Technology and the University of Gothenburg, Sweden, have proposed a way to create a completely new source of radiation. Ultra-intense light pulses consist of the motion of a single wave and can be described as a tsunami of light. The strong wave can be used to study interactions between matter and light in a unique way. Their research is now published in the scientific journal Physical Review Letters.

"This source of radiation lets us look at reality through a new angle — it is like twisting a mirror and discovering something completely different," says Illia Thiele, a theoretical physicist at Chalmers University of Technology.

Together with Dr Evangelos Siminos at the University of Gothenburg, and Tünde Fülöp, Professor of Physics at Chalmers, Illia Thiele now presents a theoretical method for creating the fastest possible single wave motion. This kind of radiation has never yet been observed in the universe or even the lab.

The radiation source is interesting for understanding the properties of different materials. Since it offers an ultra-fast switching of light matter interactions, it can be useful in material science, or sensor related research, for example. Moreover, it can be used as a driver for other types of radiation, and to push the limits of how short a light pulse could be.

"An ultra-intense pulse is like a great tsunami of light. The wave can pull an electron out of an atom, accelerating it to almost the speed of light, creating exotic quantum states. This is the fastest and strongest switch possible, and it paves the way for advances in fundamental research," says Dr Illia Thiele.

The new pulses can be used to probe and control matter in unique ways. While other light pulses, with multiple wave periods, impose changes in the material properties gradually, pulses with a single strong wave period cause sudden and unexpected reactions.

Researchers worldwide have tried to create this source of radiation, since it is of high interest for the scientific communities within physics and material science.

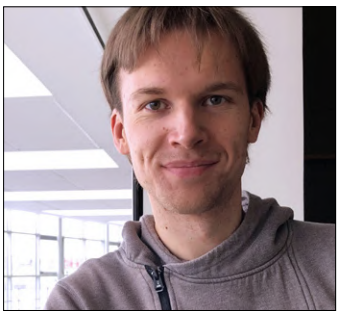
"Now, we hope to be able to bring our theoretical setup to the lab. Our method could help close the existing gaps in the scientific landscape of light sources." says Tünde Fülöp.

Read the scientific paper Electron beam driven generation of frequency-tunable isolated relativistic sub-cycle pulses in *Physical Review Letters* [1].

A more detailed description of the new method for creating ultra-intense light pulses

The researchers propose a method for the generation of ultra-intense light pulses containing less than a single oscillation of the electromagnetic field. These so-called sub-cycle pulses can be used to probe and control matter in unique ways. Conventional methods can only produce sub-cycle pulses of limited field strength: above a certain threshold the amplifying medium would be ionized by the intense fields. The researchers propose to use an electron beam in a plasma, which is not subject to a damage threshold, as an amplifying medium for a seed electromagnetic pulse. To ensure that energy is transferred from the electron beam to the pulse in such a way that a sub-cycle pulse is produced, the beam needs to be introduced at an appropriate phase of the oscillation of the electromagnetic field. This can be achieved by using a mirror to reflect the seed pulse while the electron beam is being injected. This scenario leads to significant amplification of the seed pulse and the formation of an intense, isolated, sub-cycle pulse. Readily available terahertz seed pulses and electron bunches from laser-plasma accelerators could generate mid-infrared sub-cycle pulses with millijoule-level energies, which are highly desirable as probes of matter but not possible to produce with conventional sources. ⏮

190252-01



Illia Thiele, postdoctoral researcher, Department of Physics, Chalmers University of Technology.
Photo: Mia Halleröd Palmgren/ Chalmers University of Technology.



Tünde Fülöp, Professor of Physics at Chalmers University of Technology.
Photo: Johan Bodell/Chalmers University of Technology.

Weblink

[1] Electron Beam Driven Generation of Frequency-Tunable Isolated Relativistic Subcycle Pulses : https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.122.104803

Qt for the Raspberry Pi

A step by step guide to set up the GUI framework



The Qt framework logo (www.qt.io).

By **Mathias Claußen** (Elektor Labs)

Elektor software developer and lab worker Mathias Claußen was tasked to develop a control for the FM radio on the Raspberry Pi [1] using a touch screen. The platform-independent (GUI) framework Qt has almost become the go-to industry standard to build such interfaces, so the choice was clear. For any developers out there facing similar challenges, Mathias has put together a manual that shows how set up a cross-compiler environment on a PC to allow you to compile Qt applications for the Raspberry Pi from a PC — he also includes a simple demo of course.

In software development, the client usually supplies a specification sheet — listing what functions the software should perform in the finished product. In this case, the instructions from the editorial board here at Elektor was pretty basic — "Make it so…"

Fair enough but before any work can begin it's always helpful to get a basic specification of what expected from the software. It turned out that what was needed was a Linux application running on a Raspberry Pi to provide control for some FM radio hardware and also to display RDS information if it is available in the received signal. It goes without saying that it needs to be finished by… yesterday.

At this point, I chose to implement the project using the Qt framework [2]. Qt is available under a dual-licensing model, as the source code will be *open source*, I did not need to decide between GPL and commercial versions of Qt. Using the GPL version, you can even build applications that don't use open source code — it is advisable to study the terms of the license agreement for Qt more closely. The commercial Qt license includes additional tools useful for embedded applications.

Qt

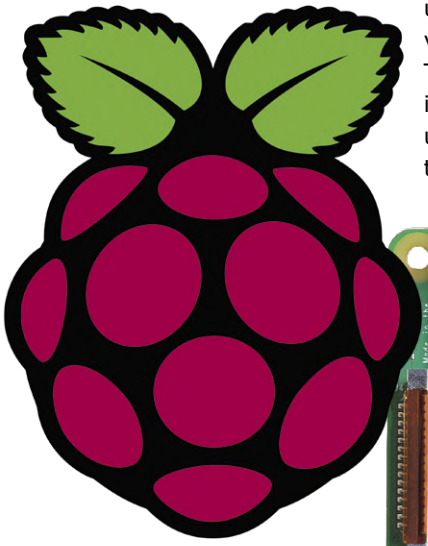
Qt [2] is a cross-platform framework that can be used to create applications in C ++/QML that ideally run on multiple platforms based on common code. This includes the usual suspects such as Windows, Linux, Android, iOS, QNX and some embedded Linux variants. Using this framework saves work in a number of places since you don't have to bother about the peculiarities of different operating systems. In addition to the many other libraries available in the framework, those designed for GUI use are especially helpful for speedy code development. What is generally true of the framework is also applicable to

the Raspberry Pi; my own experience based on the use of Qt for some previous projects (from a console application up to a small GUI tool) has demonstrated its usefulness. The elegant thing about Qt is that the experience gained in developing various projects say for the RPi is transferrable to any of the other supported platforms.

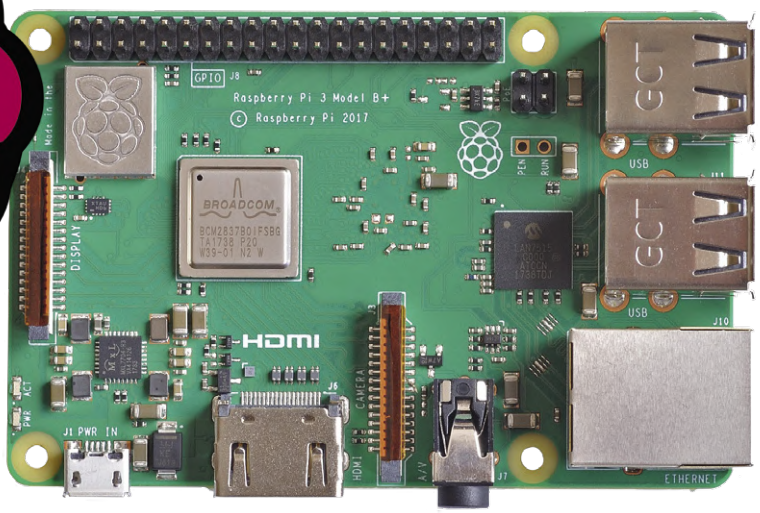
A fresher Framework

Whole books could be devoted to describing the Qt framework. This applies to many aspects of the system such as operation of the tools, development of applications and the design of graphical user interfaces. You could begin by installing a ready-made version of Qt directly on the RPi environment. This may seem like the obvious path to take but this version of Qt is now over two years old. In the world of software that time span roughly equates to two millennia. The alternative is the classic CIY (Compile It Yourself). With the benefits of all the latest security updates and patches it is always preferable, where possible, to use the most up to date version available.

The Qt framework also includes Qt creator. This useful tool just gets better with each new version. Earlier versions have proved to be a little buggy. Qt Creator also runs on the RPi, but RAM and access to the display is limited and the version is now quite old. You can build a Qt environment yourself using the latest PC version. This will allow you to develop RPi applications that do not use specialities of the RPi hardware to directly compile on a PC and test it there. With just one click and the right settings, you can then copy the software that you developed and tested on a PC to the RPi and run it there. This works well and allows remote code debugging, breakpoint setting and display of program variables.



The Raspberry Pi Foundation logo (www.raspberrypi.org).



Instructions (to code Nirvana)

Online tutorials and You Tube 'how-to's are where many people start when considering installing a new programming environment. This path can often be frustrating if you get stuck somewhere, because a step has been glossed over or maybe you've missed one setting or suddenly realise that the tutorial was written for an earlier version of RPi's operating system and gives problems with the latest version. To make things a whole lot easier I've written a manual [3] describing the steps necessary to set up Qt for the Raspberry Pi running in a cross-compiler development environment on a PC.

Since the latter is easier running under Linux, the Ubuntu PC Operating system is used. Don't worry: you won't need to buy an extra PC just for RPi development (the problem here is often not a question of the additional expense but more about desk space availability). All you need is to run a virtual machine (VirtualBox) on your PC which will use up about 60 GB of space on your hard drive.

In addition to setting up the environment, a small demo program is also described as an example to test the setup. You can also load the FM radio project into Qt and see how the different parts work together. This guide should be a starting point and show how to work with all the various tools — without needing to worry about and the setup of important details.

180477-02

@ WWW.ELEKTOR.COM

→ FM Radio HAT PCB for the RPi (160520-1 V2.0)
www.elektor.com/rpi-receiver-160520-1

→ Raspberry Pi 3 B+
www.elektor.com/rpi-3b-plus

Web Links

[1] FM Radio with RDS, Elektor 1 / 2019: www.elektormagazine.com/magazine/elektor-70/42360

[2] Qt framework: www.qt.io

[3] Qt manual (PDF, 41 pages): https://github.com/ElektorLabs/160520-QT-compile-and-install-manual

Elektor Start-up Challenge Paris 2019

An international launch pad for start-ups

By **Denis Meyer**
and **Mariline Thiebaut-Brodier**

Don't wait for Elon Musk to invite you aboard SpaceX!

After Germany and Holland, this year Elektor makes a stop in Paris with a new contest for young companies who are active in the field of embedded electronics.
This will take place at the new Forum de l'Électronique (Electronics Forum) exhibition in September 2019.
This contest, called the Elektor Start-up Challenge, aims to stimulate the advancement of young entrepreneurs after their promising debuts.
The winners will be chosen on the industrial viability of their projects, their usefulness and of course their originality.
the Elektor Start-up Challenge is the result of our close cooperation with Cap'tronic, the support program for SMEs in hardware electronic systems and embedded software, and GL Events Exhibitions, the organiser of the exhibition.

Who can enter?
Anyone who has developed a product, a software or a service in the electronics field. The **Elektor Start-up Challenge** gives you the opportunity to come and present your project to professionals in the industry.
Previous contests organised by Elektor, notably those at **electronica** in Munich, the largest electronic show in the world, have rewarded some remarkable projects in various areas of electronics.

Why get aboard this rocket?
If you enter the **Elektor Start-up Challenge** in Paris in 2019 and your application is successful, you will be welcomed with the other entrants to the **Elektor Start-up Challenge**, a large central stand at the **Forum de l'Électronique** (Paris Expo - Porte de Versailles, from the 24th to the 26th September 2019).

You will be able to:
- Present your project there in your own space
- Compete alongside other finalists
- Attract the attention of potential partners

Before, during and after the show you will achieve a high visibility:

- You and your start-up will have all the support of the **Forum de l'Électronique**: show catalogue, list of exhibitors, official site...
- You and your start-up will be presented to all Elektor magazine's readers via all our media (print, Internet, social media, YouTube channel...) in France, French-speaking countries and all over the world
- You and your start-up will be introduced to the members of the Cap'tronic program

For you and your start-up, the ultimate goal is to win the jackpot. The winner of the **Elektor Start-up Challenge** in Paris will receive an entry ticket to the next **electronica** show in Munich: you will be automatically selected to enter in the international **Fast Forward Award** contest, in the course of which you will **present your project on the stand** and benefit from the huge publicity given to this event.

The **Elektor Start-up Challenge** is organised in collaboration with:

CAP'TRONIC Systèmes électroniques et logiciel embarqué

Forum de l'électronique
Salon de l'innovation et des solutions électroniques

Making your application

- To participate in the **Elektor Start-up Challenge** at the **Forum de l'Électronique**, entrants can contact us directly at email *redaction@elektor.fr*. They can also create a (free) account at our website: *www.elektormagazine.fr*.
- You should provide the most complete documentation that you can, so that the jury can clearly appreciate the objective of your start-up, the technology used, get an idea of the product, and, if applicable, the services provided. All entries will be evaluated by the jury of the **Elektor Start-up Challenge**. To be considered a start-up, your business must have been created at most five years ago.
- If your start-up is selected to enter the **Elektor Start-up Challenge**, the representative of the business accepts to be personally present in Paris during the **Forum de l'Électronique 2019**.
- All the general conditions of entry to the **Elektor Start-up Challenge** in Paris are available online.





Participants and winners of the **Fast Forward** electronica 2018 contest.

This is where your start-up meets the future

Where? Paris Expo Porte de Versailles

When? From the 24th tot the 26th September 2019

How? Enter by Wednesday 31st July at the latest

Go to:
www.elektormagazine.fr/escparis2019



The winner will automatically participate in the **electronica** contest in Munich next year.



The Jury at work, headed by Clemens Valens from Elektor Labs.

Note:

The final date of entry is fixed at Wednesday 31st July 2019.

In case of a high number of applications, it is possible that the limited number of places will cause us to bring forward the closing date for entries. So don't wait to go to **www.elektormagazine.fr/escparis2019** to enter.

The general conditions of entry to the **Elektor Start-up Challenge** in Paris are available online.



The initial meeting between Elektor and its partners.

190232-02

New CPLD Board with MAX 10

By **Alfred Rosenkränzer** (Germany)

Upgrading the CPLD board: With an MAX10M08 CPLD as the core component, the board not only provides more logic elements than the previous version but also supports typical FPGA functions such as PLLs, ADCs and implementable microcontrollers.

In the May & June 2018 issue of Elektor magazine the author presented a board with a CPLD from the Altera (now Intel) MAX II family [1]. The EPM570 used in that project is a CPLD with 570 logic elements (LEs) in a TQFP100 package. The smaller but pin-compatible EPM240 can also be fitted on the CPLD board. Thanks to the relatively small device package, it was possible to design a prototype board that allows the IC to be used in more or less the same way as a DIL component. Although a considerable amount of logic can be built with 570 LEs, you have to switch to a different CPLD family if that is not enough. As the MAX V family only offers a few extra features compared to the MAX II family, the author decided to go directly to Intel's newest CPLD family:

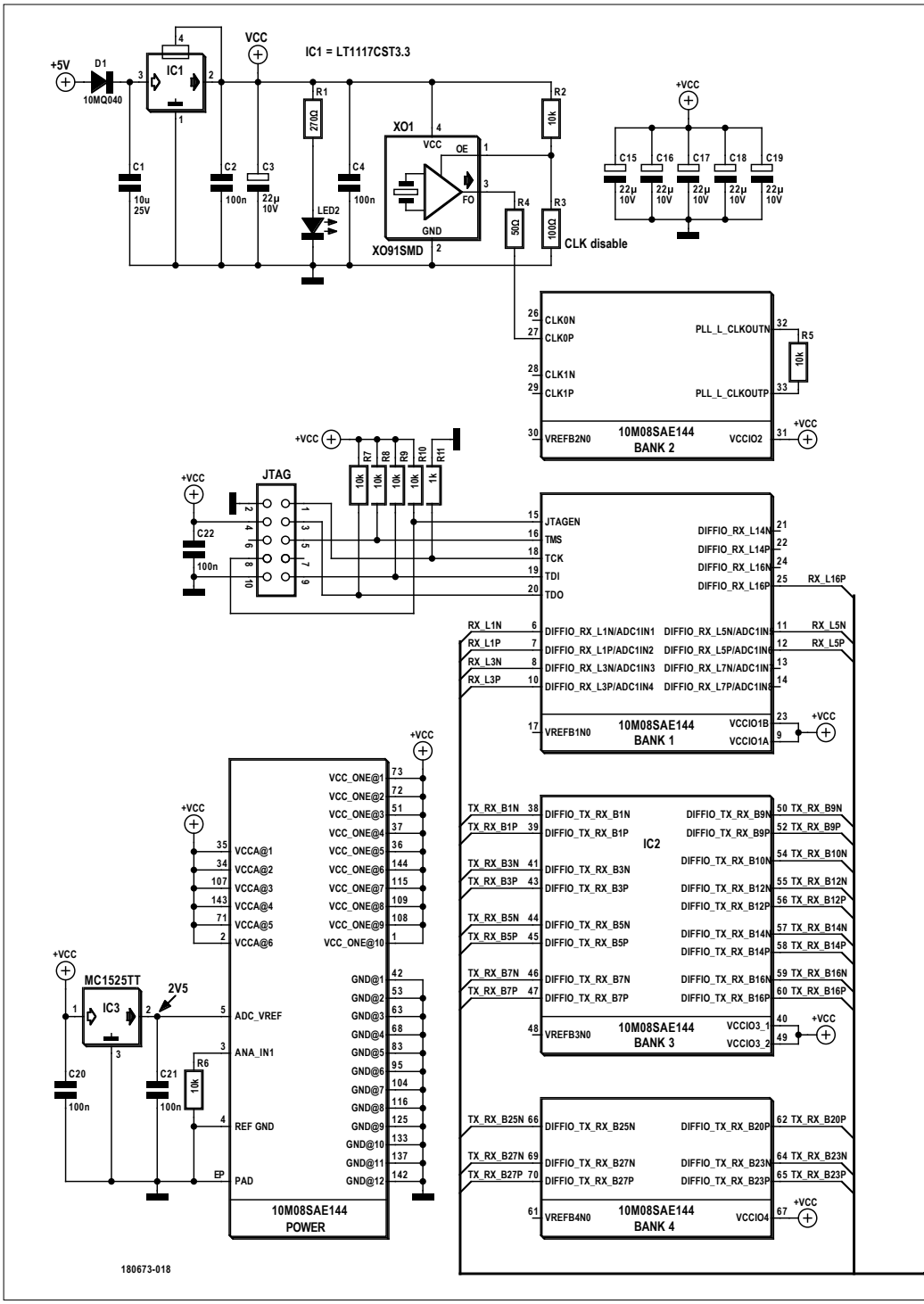
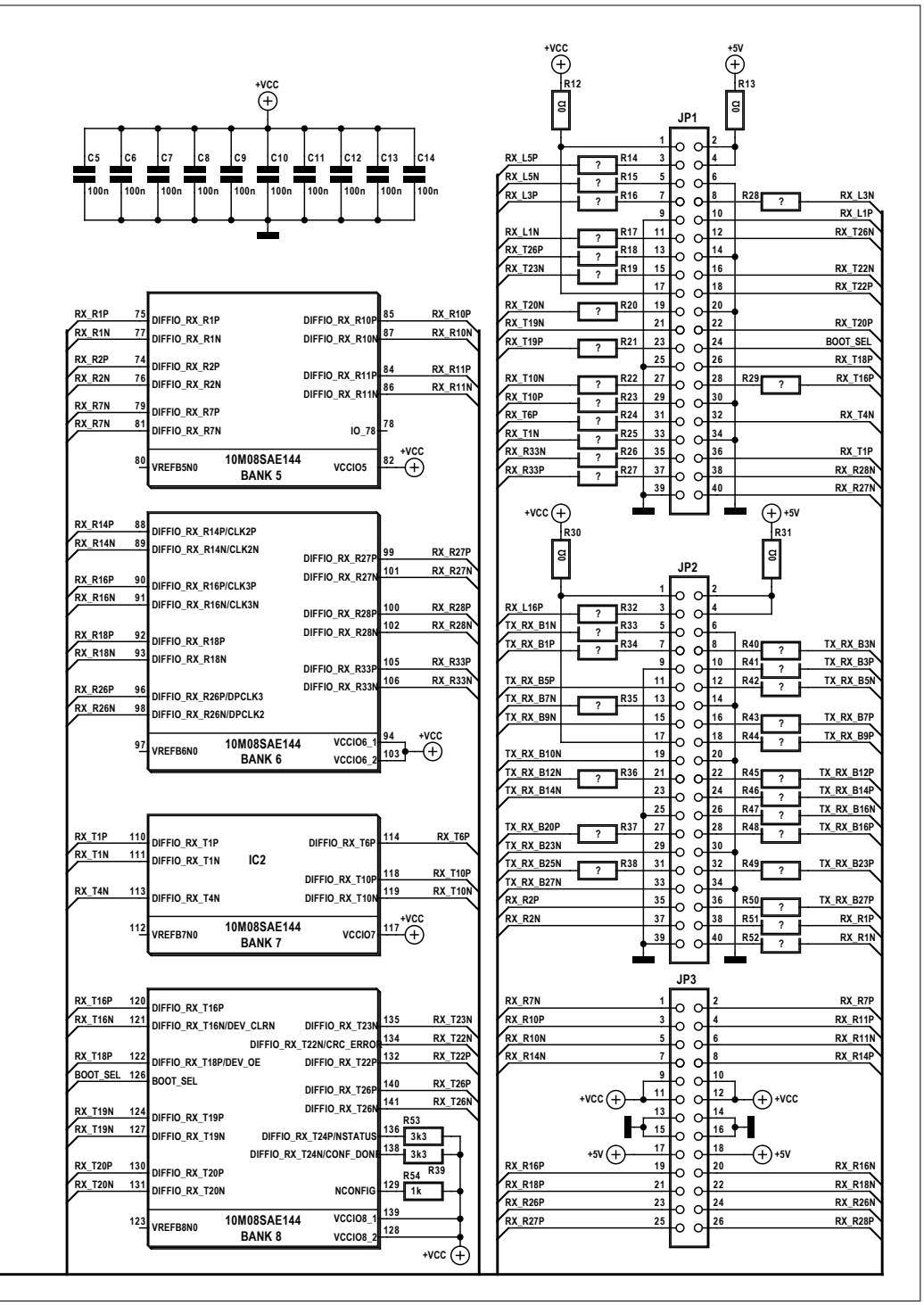


Figure 1: For ease of understanding, the MAX-10M08SAE144 is divided into many sections in the schematic diagram.

More logic with the MAX 10 family



MAX 10. This family [2] consists of seven CPLD product groups, each with several versions. You can choose from 2 K to 50 K logic elements, single or dual power supply with an integrated 3.3-V LDO voltage regulator (only single power supply is possible with the selected device), several flash memory block sizes, analog to digital converters (ADCs), temperature sensors, dual configuration flash (two dynamically usable images on the same device), and much more.

From CPLD to FPGA

In addition to pure CPLD functions, the MAX 10 family has fully operable FPGA functions such as up to four internal PLLs for generation of higher clock frequencies, implementable (NIOS) microcontrollers, digital signal processing (DSP) blocks, and soft DDR3 memory controllers.

All these different versions are available in a variety of package types, ranging from small wafer packages to large packages with hundreds of I/O pins. This is also the first selection criterion: the only package suitable for soldering by hobbyists (which means with a soldering iron) is the EQFP package with 144 pins. Although the size of this package (22 × 22 mm) makes the board somewhat larger than the previous version, it offers 101 usable I/O pins with all device options. For more detailed information, see the user guides for the MAX 10 family [3].

Peripherals

Like its smaller predecessor, this board has several peripherals in addition to the new CPLD (see the schematic diagram in **Figure 1**): a voltage regulator with decoupling capacitors, a clock oscillator, and a JTAG connector for programming. There is also an onboard voltage refe-

Pin	JP2		JP1		Rpi GPIO		Pin	Pin	JP3		Pin
1	+3.3 V	+5 V	+3.3 V	+5 V	+3.3 V	+5 V	2	1	P81	P79	2
3	P25	+5 V	P12	+5 V	GPIO 2	+5 V	4	3	P85	P84	4
5	P38	GND	P11	GND	GPIO 3	GND	6	5	P87	P86	6
7	P39	P41	P10	P8	GPIO 4	GPIO 14	8	7	P89	P88	8
9	GND	P43	GND	P7	GND	GPIO 15	10	9	+3.3 V	+3.3 V	10
11	P45	P44	P6	P141	GPIO 17	GPIO 18	12	11	+3.3 V	+3.3 V	12
13	P46	GND	P140	GND	GPIO 27	GND	14	13	GND	GND	14
15	P50	P47	P135	P134	GPIO 22	GPIO 23	16	15	GND	GND	16
17	+3.3 V	P52	+3.3 V	P132	+3.3 V	GPIO 24	18	17	+5 V	+5 V	18
19	P54	GND	P131	GND	GPIO 10	GND	20	19	P90	P91	20
21	P55	P56	P127	P130	GPIO 9	GPIO 25	22	21	P92	P93	22
23	P57	P58	P124	P126	GPIO 11	GPIO 8	24	23	P96	P98	24
25	GND	P59	GND	P122	GND	GPIO 7	26	25	P99	P100	26
27	P62	P60	P119	P120	GPIO 0	GPIO 1	28				
29	P64	GND	P118	GND	GPIO 5	GND	30				
31	P66	P65	P114	P113	GPIO 6	GPIO 12	32				
33	P69	GND	P111	GND	GPIO 13	GND	34				
35	P74	P70	P106	P110	GPIO 19	GPIO 16	36				
37	P76	P75	P105	P102	GPIO 26	GPIO 20	38				
39	GND	P77	GND	P101	GND	GPIO 21	40				

Figure 2: Pin assignments of the three connectors on the CPLD board and the Raspberry Pi GPIO connector.

rence for the A/D converter integrated in the CPLD (bank 1A).

The prototype is fitted with a type 10M08 device, which has 8 K logic elements (LEs), one PLL and one A/D converter. However, the board is suitable for all members of the MAX 10 family with 2 K to 50 K LEs, as long as they are housed in an EQFP-144 package. The board layout has four layers to ensure reliable operation even at relatively high clock frequencies.

As with the previous version, not all possible I/Os are fed out to the connectors, and for good reason. Although the board can be plugged into a breadboard or used with a stripboard like a large DIL IC, the key consideration is that both of the long pin/socket headers have the same pin assignments as the 2×20-way GPIO connector of the Raspberry Pi (type 2 or later), so that they can be plugged directly onto the Raspberry Pi. This way the processing power and PC interfaces of the Raspberry Pi can be combined with the hardware capabilities of an advanced CPLD. The pinouts of the three connectors (with the CPLD pins) are shown in **Figure 2**, along with the Raspberry Pi GPIO connector pinout for comparison. Adjacent pairs of pins, each marked with the same background colour, form differential inputs or outputs. They can also be used in single-ended mode, just like the pins without a background colour.

Combinations

This creates a variety of options. As shown in **Figure 3**, the board can be fitted directly on a Raspberry Pi (the mounting holes also match). Connector JP1 is mounted on the bottom of the board as a socket header and mates with the GPIO connector of the Raspberry Pi. The

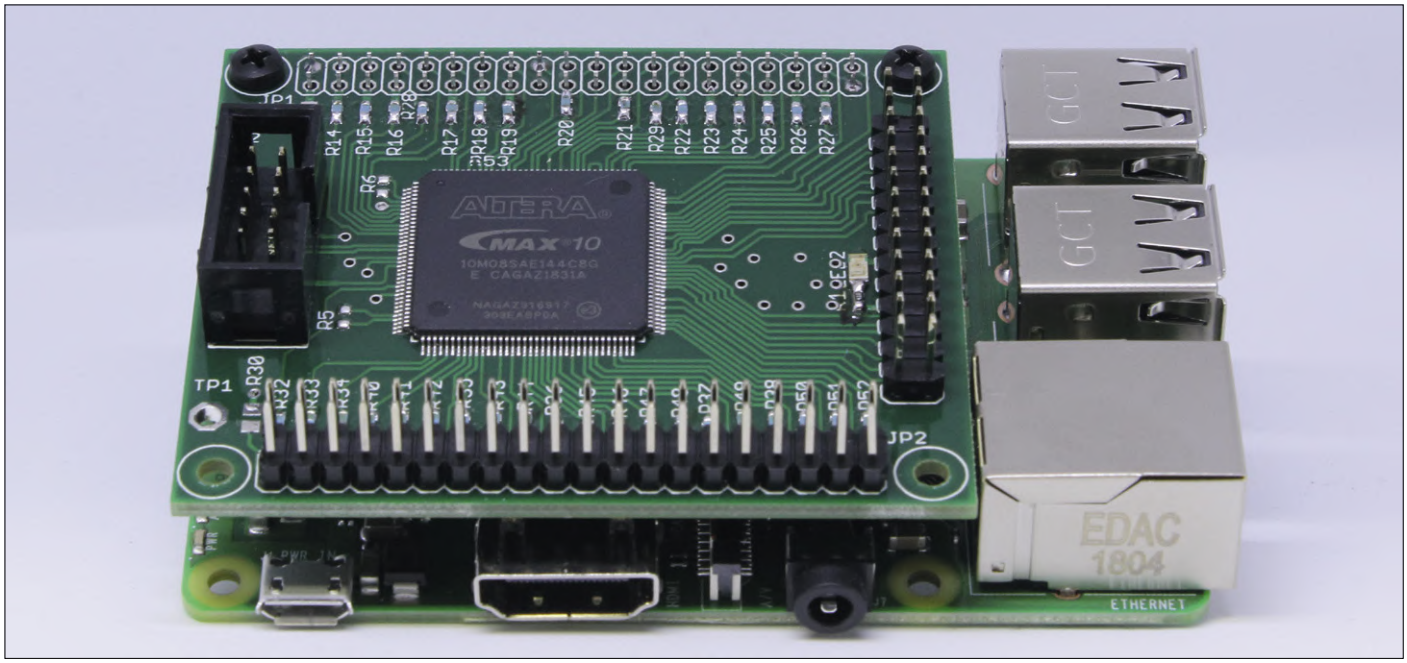


Figure 3: The CPLD board mounted on top of a Raspberry Pi 3 Model B+. The mounting holes also match.

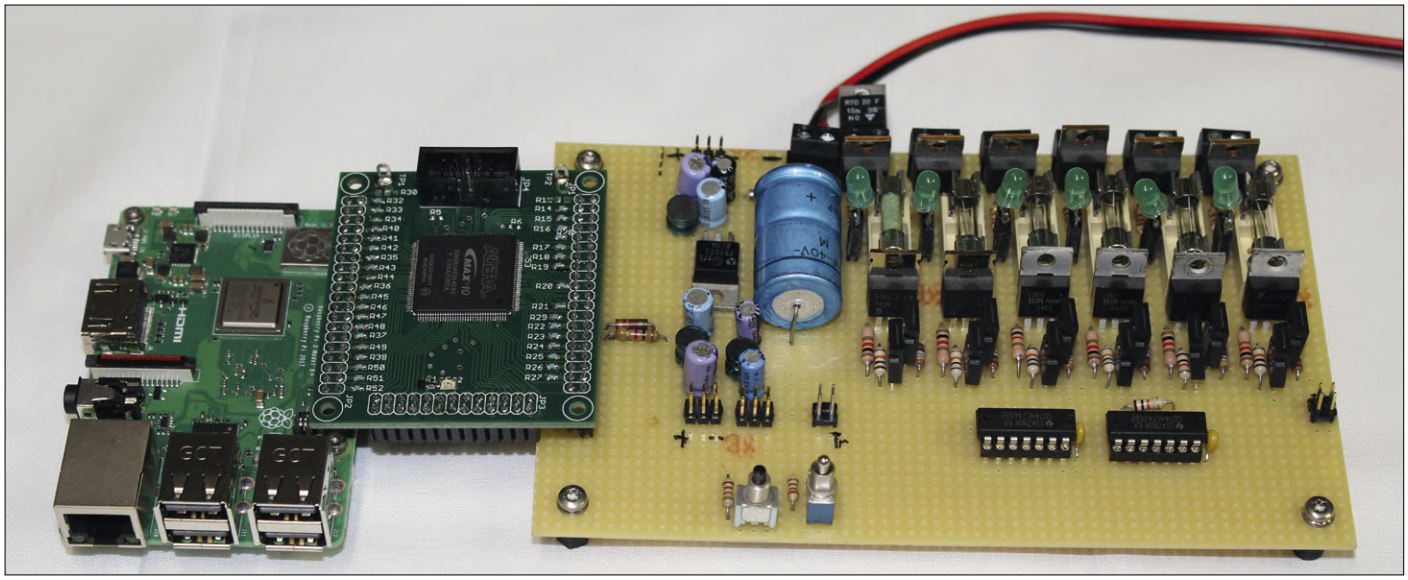


Figure 4: The Raspberry Pi controls the CPLD, which in turn controls the circuitry on the application board.

other two connectors are freely accessible, top-mounted pin headers.

Figure 4 shows how the Raspberry Pi controls the CPLD through JP2 and the CPLD in turn controls circuitry (such as a power driver) on a stripboard through connector JP1. The power supplies of the two boards can be linked or kept separate. Zero-ohm resistors (R12, R13, R30 and R31) are provided for this purpose. Connection through a ribbon cable is also possible (see **Figure 5**) if you want to put the boards in different places. Finally, you can also use the board with the 10M08SAE144 entirely on its own, without any Raspberry Pi support. ⏮

180673-02

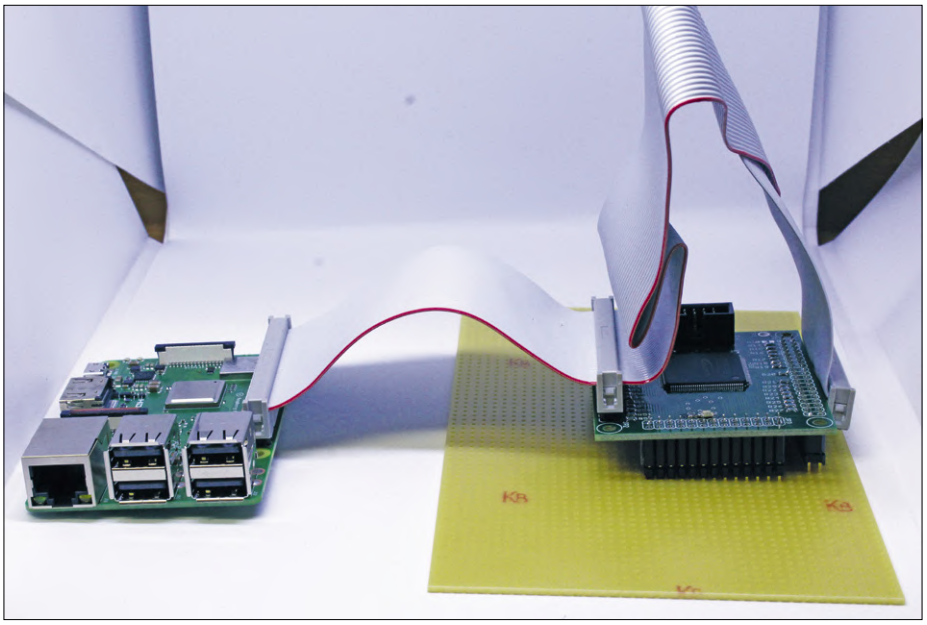


Figure 5: Distributed layout with ribbon cables.

Get it while you can

A few blank or partially populated boards (only the CPLD fitted) are available from the author.

alfred_rosenkraenzer@gmx.de

IN THE ELEKTOR STORE

→ CPLD Breakout Board in DIL Format (project from May/June 2018)
www.elektor.de/cpld-bob

Web links

[1] CPLD Breakout Board in DIL Format, Elektor May & June 2018: www.elektormagazine.com/160425

[2] MAX 10 family overview: www.intel.com/content/www/us/en/products/programmable/fpga/max-10.html

[3] Resources for the MAX 10 family: www.intel.com/content/www/us/en/programmable/products/fpga/max-series/max-10/support.html

Waterflow Monitor with ESP32

with text/SMS alert of leaks



By Denis Lafourcade (France)

There are two kinds of people: those who have already had a big water leak, and those who haven't yet had one... Insurance companies will cover this risk, but wouldn't it be better to anticipate and warn of a possible leak? The device offered here consists of a board which combines an ESP32 microcontroller for WiFi Connection and an OLED screen for local information.

Prevention is better than cure!

Everything that happens downstream of your water meter is your responsibility. In case of leaks, the extra consumption is for your account, and it can cost you dearly:

- a dripping tap: 4 l/h, that's 35 m^3 per year, over €100;
- a leaking toilet flush: 25 l/h, that's 220 m^3 per year, nearly €1000;
- a leak in the buried pipes between the meter and your house: several m^3 per hour, several thousand a week!

These figures will obviously depend on the country you live in and your cost of water.

The water companies have started to install remote meter reading systems, which include an alert if a water leak is suspected. But the deployment of these systems is slow, and most communities have not got them yet. But some water meters are equipped with a pulse transmitter which sends a pulse for each litre used, and some meters can be retrofitted with the transmitter. The system presented here uses this possibility and allows you to keep a permanent monitor on your water consumption.

Over a few months, three of my neighbours here in France discovered, after receipt of their water bill, that they had a leak between their meter and their house. The leak was undetectable for several weeks, entailing a huge overconsumption and a bill of several thousand euros!

Wishing to avoid such a disagreeable situation myself, I set out to find solutions to oversee my water consumption. By chance many water meters, mine in particular, are equipped with counter wheels. The units wheel has a magnet attached on the zero digit, allowing you to detect each revolution, which generally corresponds to a litre of use. There are on the market pulse transmitters **(Figure 1)** which clip onto the meter and produce a signal over radio or wires for each passing of the magnet past a Hall effect detector. These transmitters are powered by a Lithium battery with a life of around 15 years, so they don't need an external power source.

Wired or wireless?

I first looked at the transmission of information between the transmitter and the intelligent part of the system. The wireless pulse transmitter seemed attractive,



Figure 1. Example of a pulse transmitter in situ.

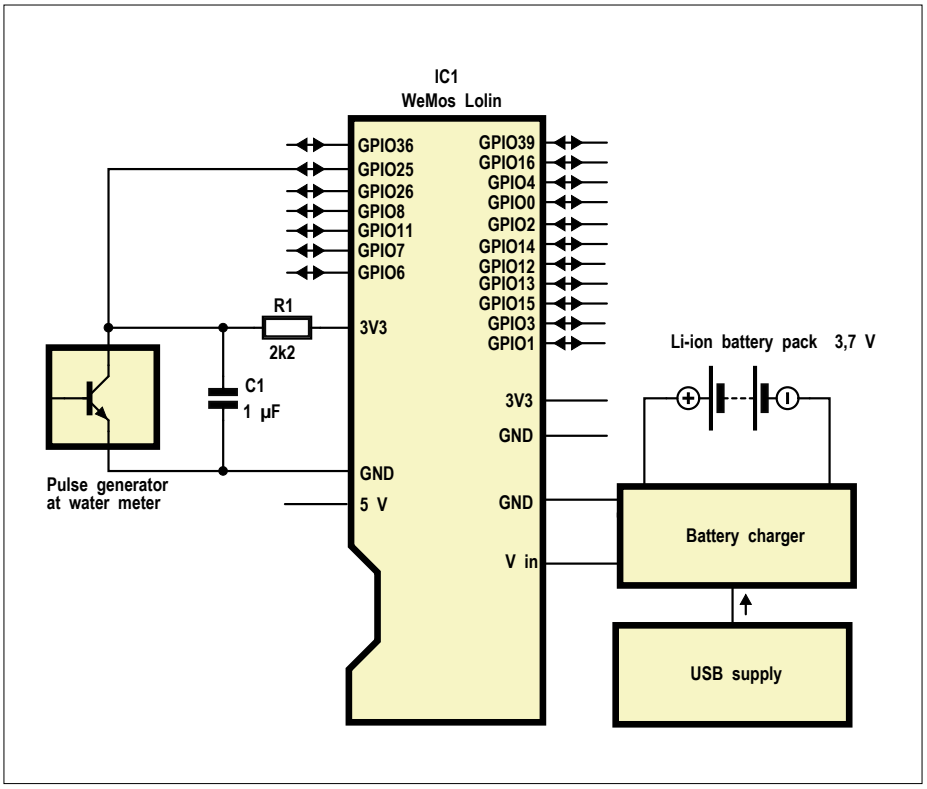


Figure 2. Circuit diagram.

but unfortunately the protocol used is not published, and in spite of considerable research, I was not able to find any useful information on the subject. If you have any better knowledge, please let me know. As my water meter and my telephone line entry are in close proximity, I was able to pull a 1-pair cable between the transmitter and the interior of my house through the telephone cable conduit. For other installations, if the provider has left the telephone cable in place after the installation of an optical fibre, I used one pair of that cable and avoided having to run a new one.

General principles

Before designing the system, it's necessary to set down the specifications:

- sensing and processing of the metered impulses;
- display of the hourly, daily and total consumption;
- algorithm for detection of suspected leaks;
- power supply with backup;
- data storage in non-volatile memory;
- network access via WiFi and time synchronisation by NTP;
- web server for remote viewing and control;
- Text (SMS) alerting system.

Electronic construction

These specifications led to the architecture illustrated in **Figure 2**. The heart of the circuit is a Lolin module made by WeMos, consisting of an ESP-WROOM-32 System-on-Chip (SoC) and a 128 × 64 pixels OLED screen (connections in **Figure 3**). This board is available in several version on the Internet, for a few dollars, and also in the Elektor Store. The ESP32 has WiFi and sufficient memory capacity for practical applications. As well as this, it is compatible with the Arduino IDE, which makes it partic-

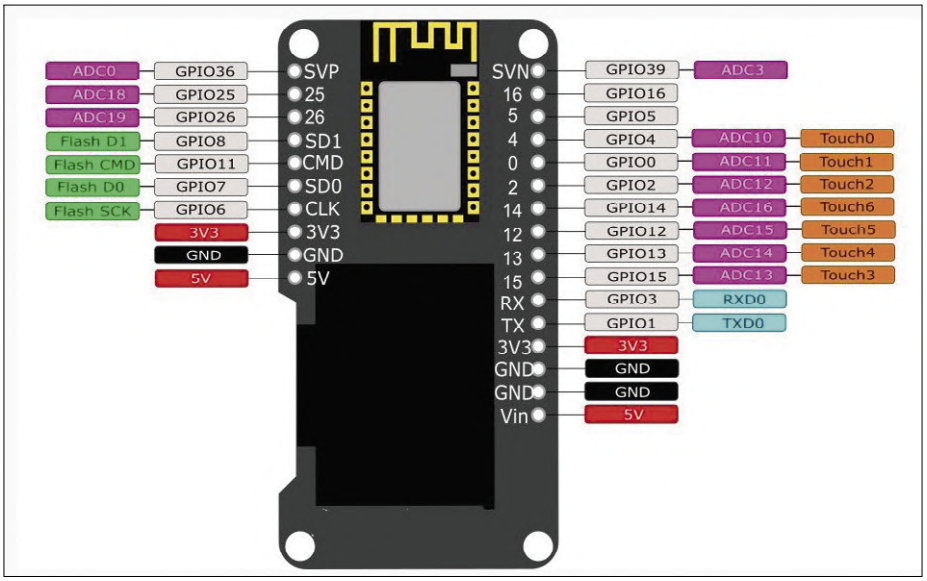


Figure 3. Connections of the WeMos Lolin module.

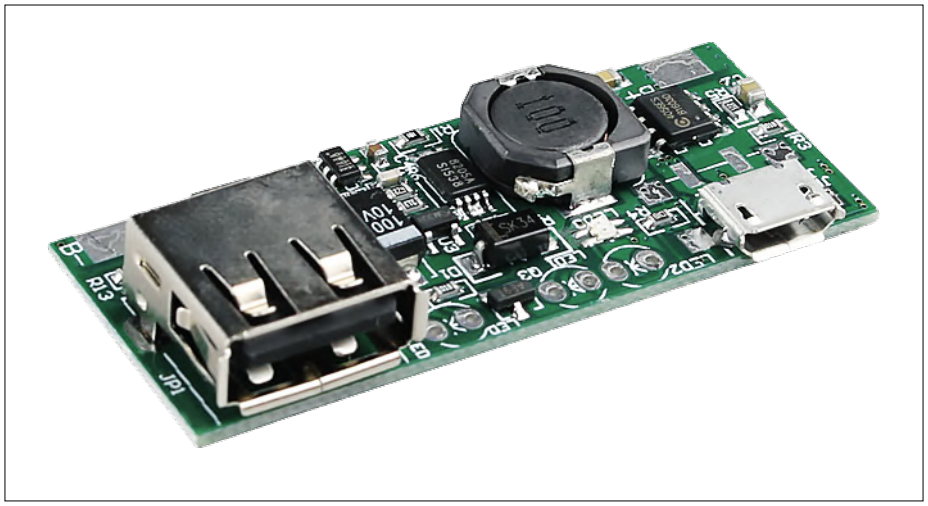


Figure 4. Power supply — battery charger module.

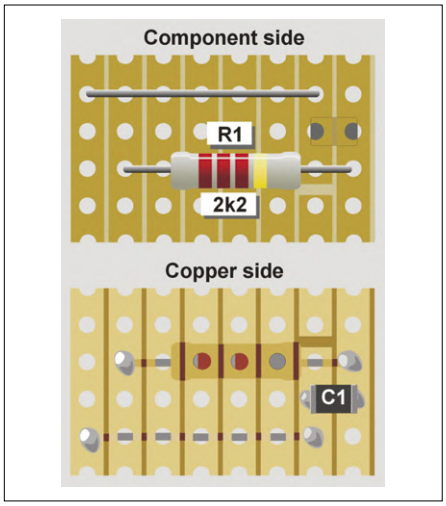


Figure 5. Interface board (connected between the GND and GPIO25 pins of the Lolin module).

ularly simple to use. Power is secured with a Li-Ion battery with a 5 V charger from a micro-USB socket. This charger (**Figure 4**) is designed for Li-Ion batteries and maintains the charge level and protects against excessive discharge and overcharging. It is available from several Internet sites (Search "3.7V Li-ion Battery Mini USB To USB A Power Supply Module"). In the case of mains power failure, the battery can power the system for at least 36 hours.



Figure 6. the three pages of the OLED screen.

The pulses from the transmitter are of open-collector type, with a ground pulse for every litre consumed. Acquiring these pulses is very simple: a pullup resistor between the data line and the 3.3 V pin on the board, and a 1-µF capacitor to filter out any interference on the line. The data line is connected to pin GPIO 25 of the ESP32, and the earth line to the GND pin. The connection of these two wires, the resistor and the capacitor, is made on a piece of stripboard soldered on the male pins of the module (**Figure 5**).

Software

The heart of the system is of course the software (available on the article page [2]). Here again, it is not necessary to reinvent the wheel, because many examples already exist and are made available by the community of "makers".
The following libraries provide most of the services that we need:

- `Wifi`, `ESPmDNS`, `WiFiUdp` and `Arduino OTA` for Wi-Fi access and wireless software updating;
- `TimeLib` for time setting by NTP (Network Time Protocol);
- `ESP32WebServer` and `WiFiClient` for the web server;
- `Wire`, `SSD1306` and `OLEDDisplayUi` to drive the OLED display;
- and of course the power and functionality of the `ESP32 Arduino Core`.

After the inclusion of all these libraries and the definition of the structures and constants of the program, we find the classic Arduino program architecture, that is to say an initialisation routine `setup()`and a main loop `loop()`. The `IRAM_ATTR handleInterrupt()` handles the interrupt triggered by the grounding of the data line connected to GPIO 25, and its only task is to increment the value of the counter for litres consumed. Software filtering takes care of any "bounce" on the line.

The main loop takes care of the following operations.

- **Display on the OLED screen**

The `OLEDDisplayUi` library allows us to implement simply a very elegant display: three pages are defined and they cycle every five seconds (**Figure 6**). The first page displays the date (the time is displayed at the top right of all three pages), today's consumption and the leak detection status. The second page shows the value of the counter, the value of the last reading and the consumption since then. The third page gives the status of the WiFi connection and the IP address used.

- **Web Server**

The server listens on a port defined during the initialisation, and you must thus ensure redirection on your router. When a user connects through a browser, the server presents one page with three tabs (**Figure 7**). The `Visualisation` (Overview) tab, refreshed every 5 seconds, shows, in addition to the information shown on the OLED display, an hour-by-hour table of consumption over the last 24 hours. The `Saisie` (Setup) tab allows the user to define various parameters: the initial value of the counter, offi-

cial value of the last reading, and value of the alert threshold for daily consumption (see below). Access to this tab is protected by a username and password. The `À propos` (About) tab is a reminder of the functions of the system and shows the software version.

- **Leak detection**

Every hour, the software looks at the hourly consumption and establishes a leak diagnosis on the basis of the following criteria:
- the consumption for the past 24 hours is above a threshold;
- there is not a baseline of two consecutive hours without consumption.

In the absence of a leak, there should be, notably at night, at least two consecutive hours without any consumption. If that's not the case, there's a strong possibility of a leak.
In the case of presumption of a leak, the type of leak is shown on the first page of the LED screen as well as on the web page, and an SMS is sent to a predetermined number thanks to the IFTTT service.

- **IFTTT**

(If This Then That): is a free service which combines actions triggered by events, the combination being labelled an "applet". Here the event will be the sending by our software of a specific "Post" request to the site `maker.ifttt.com` which then sends an SMS to the number defined during the creation of the "applet". In case of a leak, the software will send an SMS every hour, but as IFTTT limits the number of SMS per month, the software will not send more requests once this limit is reached.

- **Time setting and backup**

Setting the correct time is done every hour by NTP (Network Time Protocol) and the time is maintained by the real time clock (RTC) of the ESP32. Changes for Daylight Saving Time are also automatic. Every hour, the software saves the current consumption values in non-volatile memory (EPROM).

Assembly

A 3D printed case was specifically designed for containing the system (**Figures 8** and **9**). Openings have been placed for the OLED screen, cooling the SOC, and for the WiFi antenna. A cut-out motif in the form of a wave, as well as adding an aesthetic aspect, allows ventilation of the case should the battery generate any heat while charging. On the inside, the case provides fixing supports for the Lolin module and the battery. The charger is simply attached with a strip of double-sided tape.
All that is needed is to connect the cable from the pulse transmitter, then a micro-USB cable with a standard charger. A USB to micro USB cable of 10 cm connects between the Lolin module and the charger.

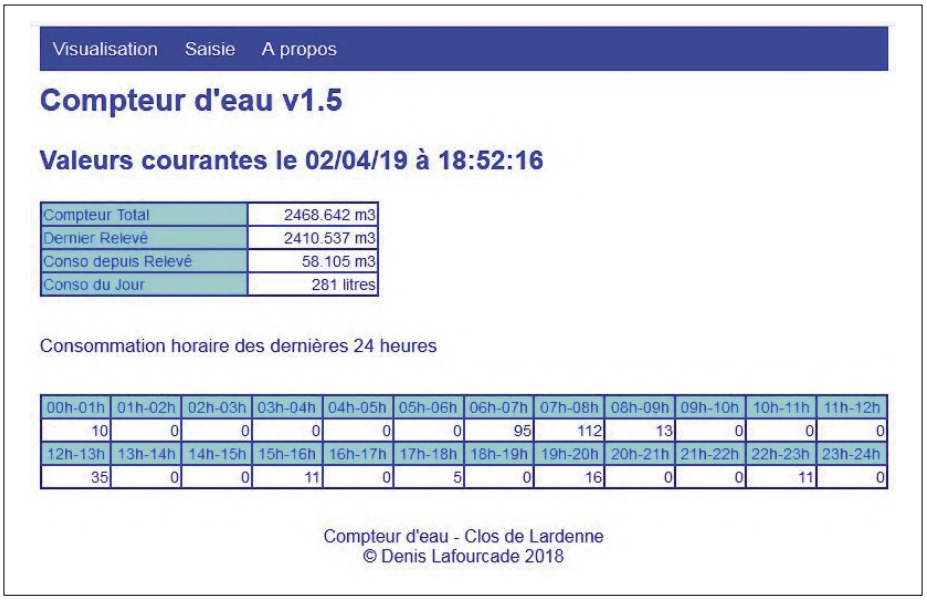
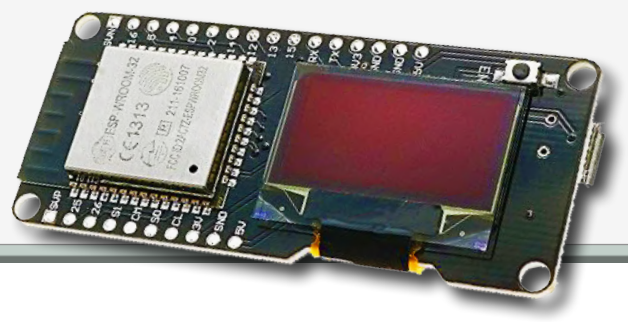


Figure 7. Web server.

Shortcuts to access the web page

Users may save shortcuts on their smartphones to access this Web page. For those whose access provider offers a router with 'loopback', a single shortcut will suffice, with the external address of their router followed by the chosen port number. For others, they will need a second shortcut with the address of the server on their Local Area Network (192.168.1.234:4321 for example). Some operators offer, by default or on demand, a fixed IP address. In that case, the external shortcut is still valid. Other operators change this address periodically, which means updating the shortcut or resorting to a Dynamic Domain Name Server (DynDNS) which are sometimes free. How to do this is outside the scope of this article, but you will find all you need for this on the internet.

@ WWW.ELEKTOR.COM
→ Wemos Lolin ESP32 Module with OLED screen
www.elektor.com/wemos-lolin



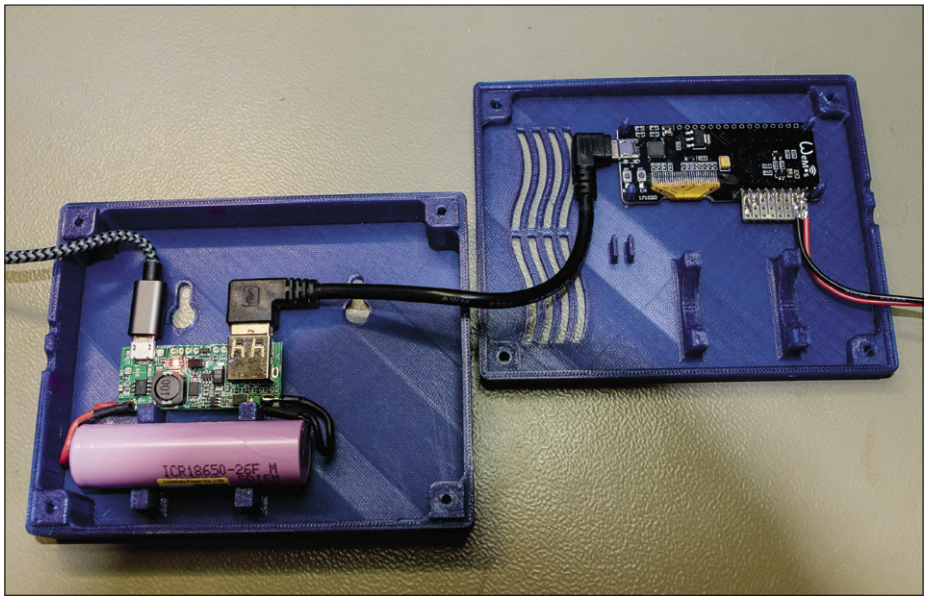


Figure 8. Photo of the complete system, installed in its case.

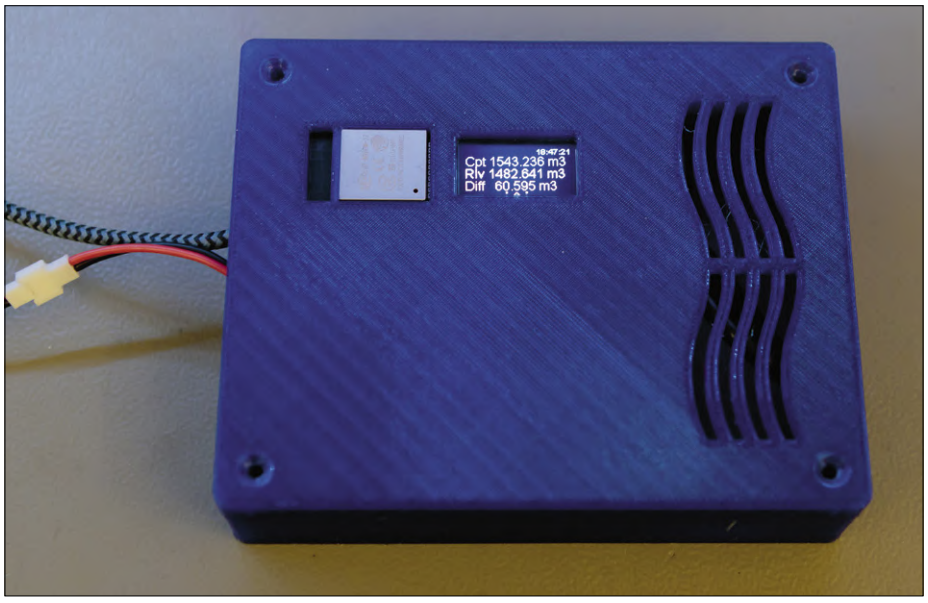


Figure 9. Finished operational unit in its 3D printed case.

Commissioning and use

Programming of the system is done with the Arduino IDE, having first configured it for ESP32-based boards and selected WeMos Lolin32 as the board type. In addition it is necessary to load all the libraries shown above. The first upload is done with the module connected to a PC, following which they may be done over WiFi thanks to the OTA (Over The Air) functionality.

The system starts up after power is applied, looks for the programmed WiFi network, sets the time and waits for a user connection while monitoring the pulses from the meter. It should then be accessed from the web page using the address on the third page of the screen, to set the actual value of the meter, that of the last reading and the consumption alert threshold.

Conclusion

The manual reading of our meters every six months shows a difference with this system of only a few litres, for consumption of 50 m3 (= 50 000 litres) which is perfectly satisfactory. several alerts were triggered and allowed early detection of leaks before they became serious.

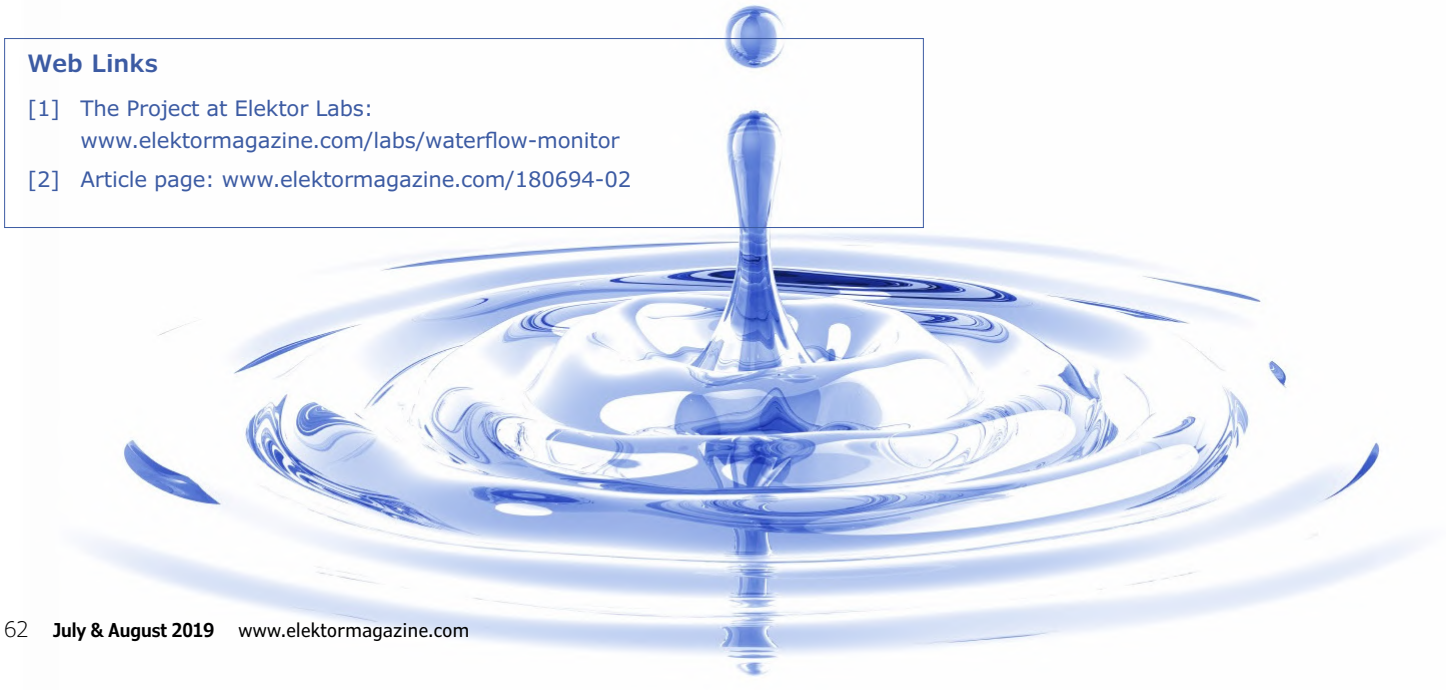
I have installed several of these systems for my neighbours, and all of them reported that they used them profitably to monitor their consumption and even optimise it by noting the consumption for a shower, washing machine, dishwasher, refilling the pool, etc. This is another interesting application not originally foreseen! Please share your own experiences! ⏮

(180694-02)

Web Links

[1] The Project at Elektor Labs: www.elektormagazine.com/labs/waterflow-monitor

[2] Article page: www.elektormagazine.com/180694-02



TL431 Precision Programmable Reference

Peculiar parts, the series

By **Neil Gruending** (Canada)

There's a good chance that you've used a TL431 programmable reference recently without even knowing it. These small 3-terminal devices blend in with their surrounding circuitry which makes them difficult to spot but they are the voltage reference used in many power supplies. Let's see what makes them so popular.

The Texas Instruments TL431 was originally released in 1978 as an improved version of the TL430 shunt regulator. The TL431 had better voltage accuracy and stability so it was called a 'precision adjustable shunt regulator'. It was also designed to have sharper turn-on characteristics compared to temperature compensated Zener diodes.

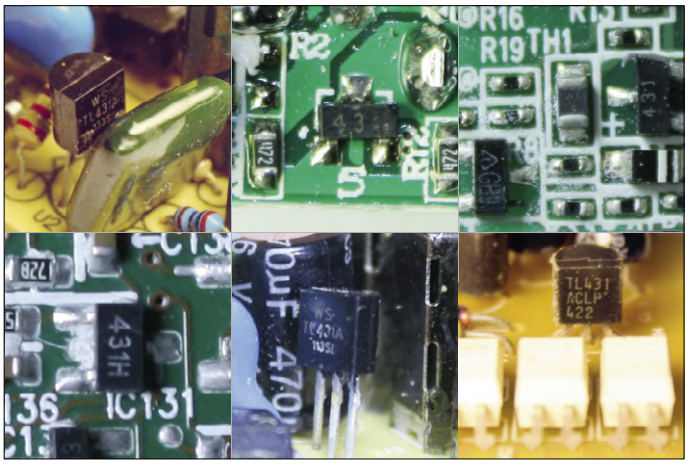
A Texas Instruments TL431 incorporates a bandgap voltage reference, opamp and output drive transistor like in **Figure 1** even though its schematic symbol is usually drawn as **Figure 2**. The opamp is wired with the internal 2.5-V reference on the negative input and the positive input is brought out to the Reference pin so that the amplifier can be configured externally. The transistor on the amplifier output increases the current sink capability to 100 mA. This configuration makes the TL431 a very versatile device.

For example, if you apply a voltage to the Reference input the TL431 will act like a comparator with a nice temperature compensated internal voltage reference. Or you can connect the reference input to the cathode like in **Figure 2** to get a 2.5-V output on the cathode. Add two resistors to set the amplifier gain and you can set the output voltage to any value you want.

You will find that the TL431 is used a lot in various power supplies. For example, a PC ATX power supply can use up to three of them with one used for the standby power, main power and the 3.3-V regulator. They are so common in ATX power supplies that many of the PWM controller ICs include TL431's internally to reduce the part count. Other applications include battery chargers and even a crystal radio amplifier [3].

I hope that I've inspired you to try a TL431 the next time you need a precision reference voltage source. There are plenty of great circuits available and make sure to check out Ken Shirriff's site where he reverse-engineers the TL431 [2] for some great photos of the die.

180575-D-01



Six examples of power supplies using the TL431. Top row: cheap 5-volt power supply, cheap phone charger, Apple iPhone charger (also 'GB9' variant in lower left). Bottom row: MagSafe power adapter, KMS USB charger, Dell ATX power supply (with optoisolators in front). Source: Ken Shirriff, www.righto.com/2014/05/

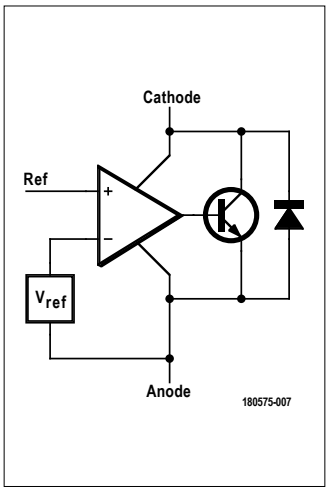


Figure 1: TL431 equivalent circuit [1].

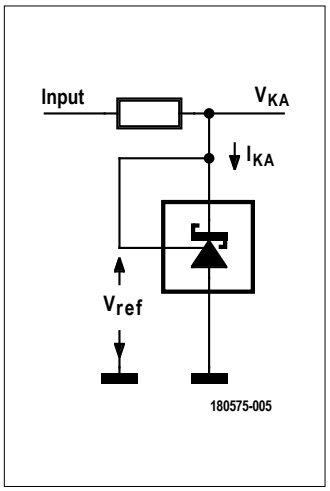


Figure 2: TL431 schematic symbol [1].

Web Links

[1] TL431 / TL432 datasheets: www.ti.com/lit/ds/symlink/tl431.pdf

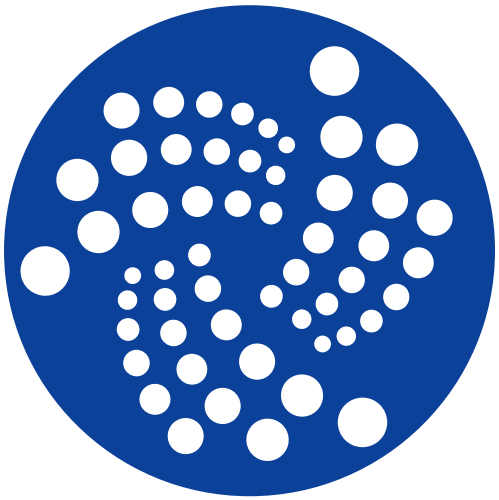
[2] TL431 reverse engineered: www.righto.com/2014/05

[3] TL431 as an amplifier: www.techlib.com/electronics/crystal.html#AudioAmp

IOTA Crypto Currency (and an FPGA for the RPi)

Part 2: PiDiver — FPGA board for fast calculations

By **Thomas Pototschnig** and **Julia Orlitzky** (Germany)



In the last issue we presented the 'IOTA' crypto currency which permits IoT nodes to pay other subscribers — for example for storing data [1]. In order to perform IOTA transactions, a proof-of-work calculation task must first be solved; this protects the transaction database ("tangle") from spam. Even x86 computers sweat during the arithmetic task, but IOTA provides for smaller IoT nodes to outsource this work to specialized devices. The second part of the series introduces such a platform based on an FPGA add-on board for the Raspberry Pi.

This installment of the article describes a fast and energy-efficient solution to the IOTA proof-of-work (PoW) problem; the necessary calculations are performed by an FPGA. This achieves an average PoW time of 300 ms, which is not only faster than an SSE-optimized multi-threaded library on a Core i5 (about 1.7 s). The power consumption of 2 watts is also significantly lower.

IOTA's proof of work

For a transaction to be accepted by the tangle, the last 14 steps [1] of the transaction hash must be zero. If a hash does not fulfil this condition, a counter (Nonce, see **Figure 1**) is incremented in the transaction and the transaction hash is recalculated. This is repeated until a valid hash is found. On average, it takes 314 / hps seconds to solve the calculation problem, where hps is the number of hashes that can be calculated per second. This results in an average of 4.78 million iterations until the problem is solved.

The idea behind this is that the tangle can be protected from spam, since the maximum achievable transaction speed is greatly reduced by the computing effort. Verification, on the other hand, takes hardly any time because only the hash of the transaction must be checked. A valid transaction hash is therefore called a proof of work because it is proof that the calculations have been performed.

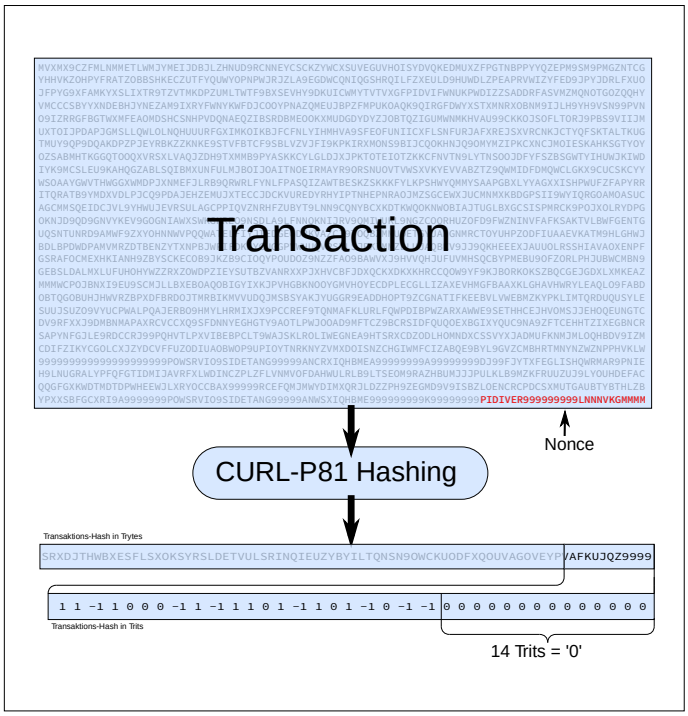


Figure 1: Transaction hash.

The hashing algorithm CURL-P81 is used, which — similar to SHA3 — is designed as a sponge construction. New blocks with a fixed block size (rate) are absorbed into the internal state, which has three times the size (capacity) of a block. The entire state is then transformed into the subsequent state using a sponge function (**Figure 2**). Once all blocks have been absorbed, the hash can be squeezed out.

An IOTA transaction consists of 33 blocks of 81 trytes each (B1–B33 in the diagram). The last block contains the counter that is incremented if the transaction hash does not fulfill the condition of the main net (there are other tangle nets for development and test).

Since the counter is only in the last of the 33 blocks, the state after absorbing the 32nd block ("mid-state") can be used as the start state for the 33rd block. After incrementing the counter in the last block, only a single block needs to be hashed in order to calculate the hash of the entire transaction, which saves a lot of time.

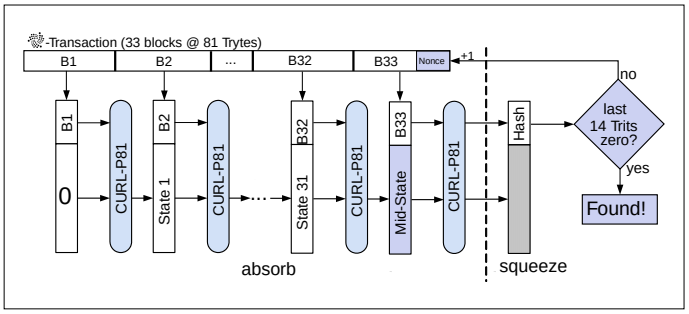


Figure 2: The sponge construction of the hash calculation (source: Wikipedia).

Overview of PiDiver

The PiDiver — the name consists of Raspberry Pi and the reference PoW algorithm "Pearl-Diver" [1]) — is a HAT (Hardware Attached on Top) for Raspberry Pis, which can perform IOTA's proof-of-work quickly and energy efficiently.

Figure 3 shows the block connection of the PiDiver. In the centre is an FPGA Cyclone 10 LP from Intel. This is a member of the relatively new Cyclone family (released 2018), with which Intel is targeting low-cost and low-power IoT applications in particular. The FPGAs, which offer a lot of logic resources, are not only very inexpensive and power-saving, but also available in (relatively) DIY friendly housing variants (EQFP), making them the perfect choice for this project.

The FPGA is clocked at 200 MHz and can calculate a complete hashing lap per clock cycle (CURL-P81 uses 81 laps). Furthermore, it calculates seven hashes with different counters (non-cents) simultaneously. In addition to the 81 clock cycles, two more clock cycles are required to check the result, restore the old state (mid-state in Figure 2) and set the new counter values. You get about 16.8 MHash/s. In the case of the IOTA Main Net, the average proof-of-work time is approximately 300 ms or 3.33 PoW/s, including data transfer.

Furthermore, there is an STM32 microcontroller (STM32F302) in the circuit which makes it possible to operate the FPGA via USB. The proof-of-work can thus be performed either via the SPI interface of a Raspberry Pi or on the PC via USB (without Raspberry Pi).

You may notice that there is no flash memory for configuring the FPGA. Consequently, the configuration is reloaded into the FPGA after each power cycle. This can also be done via the Raspberry Pi or via USB. The configuration is automatically detected and executed by the PiDiver library. For future expansion, an SPI flash memory was provided on the STM32 that is large enough to store the complete FPGA bitstream. This would allow an automatic configuration after the power-up, which could be interesting for embedded applications (for example with an ESP32 microcontroller), because the bitstream of about 800 kB is too large for the flash memory of many microcontrollers.

The circuit diagram in **Figure 4** does not offer much more than this. There are three supply voltages, each with its own voltage regulator. Two of these are low-drop fixed voltage regulators

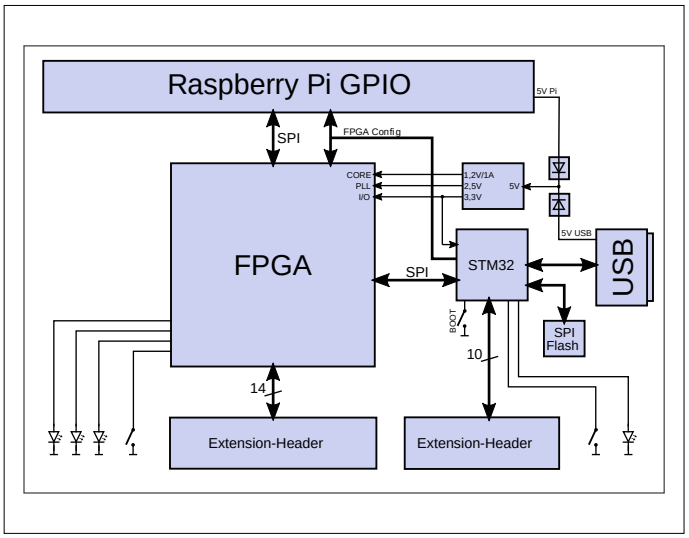
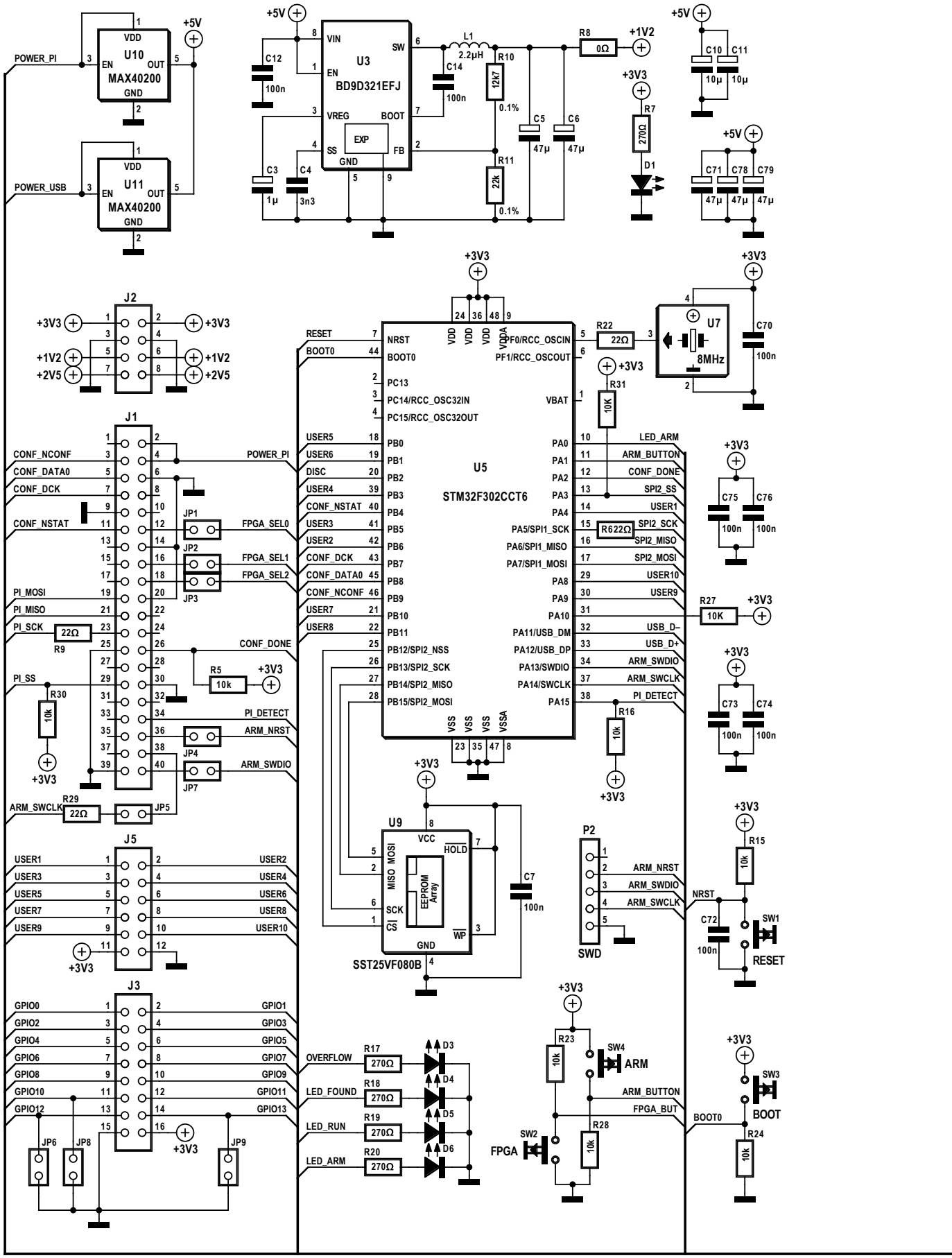


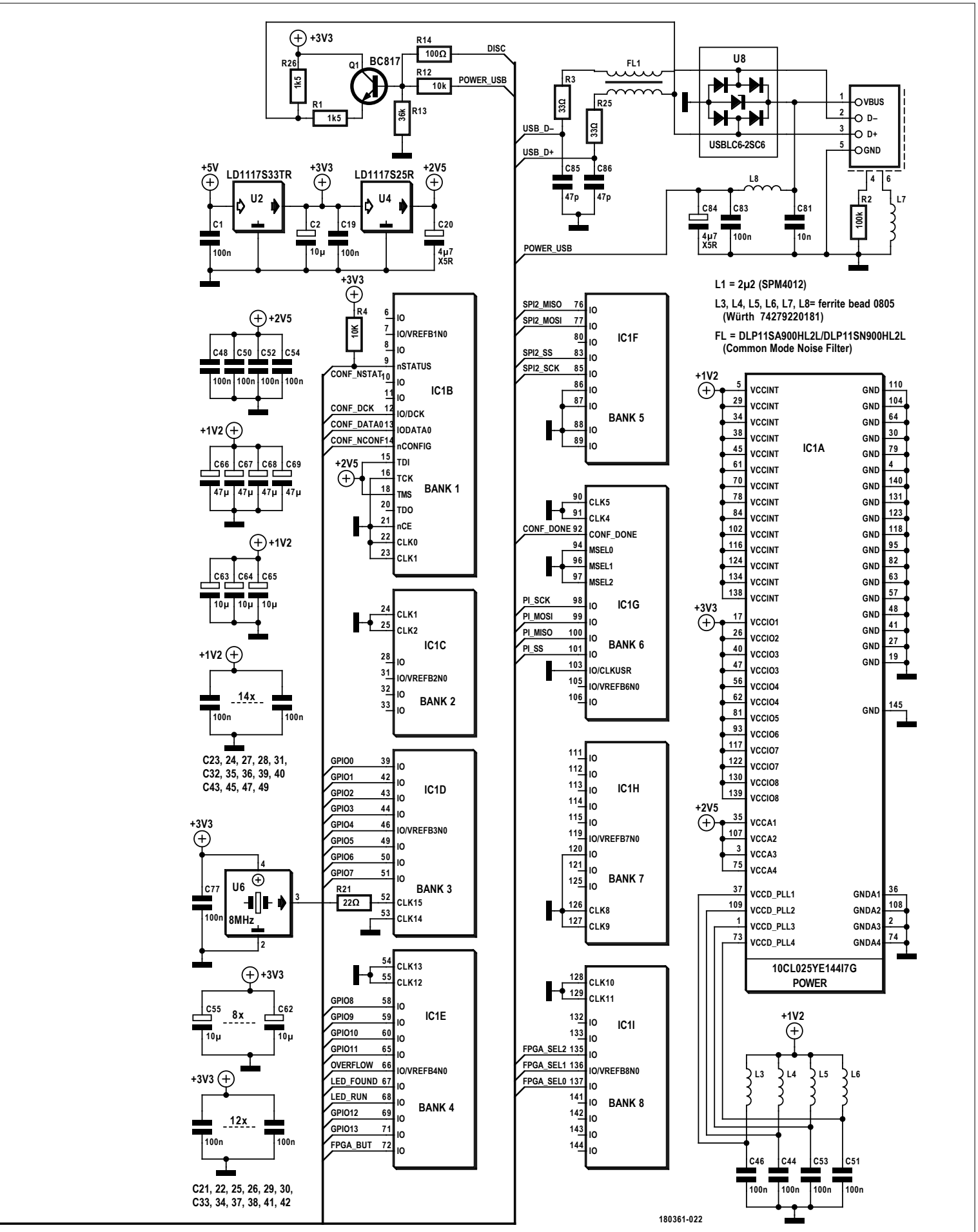
Figure 3: Block diagram of the PiDiver with the central FPGA.

(LDOs) connected in series for 3.3 V and 2.5 V, the third is a switching regulator set to 1.2 V which can supply up to 3 A current (1 A is required). The board can be supplied with 5 V either via the Raspberry GPIO connector or via USB. According to the manufacturer, two "ideal diodes" (type MAX40200) combine both 5-V voltages so that the two potential voltage sources are not directly connected to each other. The MAX40200 is in fact not a diode, but a logic MOSFET with a voltage drop of just 85 mV at 1 A. Whether this minimal power dissipation makes every Schottky diode green with envy!

The circuit shows some extension possibilities for other projects. There are two extension headers for this, on which 14 I/Os of the FPGA and ten port pins of the STM32 microcontroller for your own applications are located. There are also LEDs and buttons on the FPGA as well as on the STM32. The circuit was designed so that the STM32 internal DFU boot loader can be used. A separate BOOT button must be pressed during power-up so that the controller starts in bootloader mode. The total current consumption of 350 mA is so low that operation on a standard USB port is possible without any problems.

Figure 4: Schematic of the PiDiver.




+3V3
R14
100Ω
DISC
R26
1k5
Q1
BC817
R12
10k
POWER_USB
R1
1k5
36k
R13
+5V
LD1117S33TR
U2
+3V3
LD1117S25R
U4
+2V5
C1
100n
C2
10µ
C19
100n
C20
4µ7
X5R
+3V3
R4
10K
CONF_NSTAT
IC1B
BANK 1
CONF_DCK
CONF_DATA0
CONF_NCONF
nSTATUS
IO/DCK
IODATA0
nCONFIG
TDI
TCK
TMS
TDO
nCE
CLK0
CLK1
+2V5
C48 C50 C52 C54
100n 100n 100n 100n
+1V2
C66 C67 C68 C69
47µ 47µ 47µ 47µ
+1V2
C63 C64 C65
10µ 10µ 10µ
+1V2
14x
100n 100n
C23, 24, 27, 28, 31,
C32, 35, 36, 39, 40
C43, 45, 47, 49
IC1C
BANK 2
CLK1
CLK2
IO
IO/VREFB2N0
IO
IO
IC1D
BANK 3
GPIO0
GPIO1
GPIO2
GPIO3
GPIO4
GPIO5
GPIO6
GPIO7
IO/VREFB3N0
R21
22Ω
CLK15
CLK14
+3V3
C77
100n
U6
8MHz
+3V3
C55
8x
C62
10µ
10µ
+3V3
12x
100n 100n
C21, 22, 25, 26, 29, 30,
C33, 34, 37, 38, 41, 42
IC1E
BANK 4
CLK13
CLK12
GPIO8
GPIO9
GPIO10
GPIO11
OVERFLOW
LED_FOUND
LED_RUN
GPIO12
GPIO13
FPGA_BUT
IO/VREFB4N0
USB_D–
USB_D+
R3
33Ω
R25
33Ω
FL1
C85
47p
C86
47p
POWER_USB
U8
USBLC6-2SC6
VBUS
D–
D+
GND
L8
C84
4µ7
X5R
C83
100n
C81
10n
R2
100k
L7
L1 = 2µ2 (SPM4012)
L3, L4, L5, L6, L7, L8= ferrite bead 0805
(Würth 74279220181)
FL = DLP11SA900HL2L/DLP11SN900HL2L
(Common Mode Noise Filter)
SPI2_MISO
SPI2_MOSI
SPI2_SS
SPI2_SCK
IC1F
BANK 5
CONF_DONE
CLK5
CLK4
CONF_DONE
MSEL0
MSEL1
MSEL2
PI_SCK
PI_MOSI
PI_MISO
PI_SS
IC1G
BANK 6
IO/CLKUSR
IO/VREFB6N0
IC1H
BANK 7
IO/VREFB7N0
CLK8
CLK9
CLK10
CLK11
IC1I
BANK 8
FPGA_SEL2
FPGA_SEL1
FPGA_SEL0
IO/VREFB8N0
+1V2
IC1A
VCCINT
GND
+3V3
VCCIO1
VCCIO2
VCCIO3
VCCIO4
VCCIO5
VCCIO6
VCCIO7
VCCIO8
+2V5
VCCA1
VCCA2
VCCA3
VCCA4
VCCD_PLL1
VCCD_PLL2
VCCD_PLL3
VCCD_PLL4
GNDA1
GNDA2
GNDA3
GNDA4
10CL025YE144I7G
POWER
+1V2
L3
L4
L5
L6
C46
100n
C44
100n
C53
100n
C51
100n
180361-022

Superstructure

The PCB of the PiDiver [2] in **Figure 5** has four layers and is almost exclusively populated with SMDs. Some (U6, U7 and FL1) can't be solved with a soldering iron, because the pads are hidden under the housing. Soldering paste and hot air are therefore recommended for assembly.

You should leave R8 (a 0-Ω resistor at the output of the switching regulator U4) in its box for the moment, because this resistor is not populated at first. The components crossed out in the assembly diagram (see [2]) are not required for operation as PoW accelerators. The components R12, R13, R14, Q1 and P2 are only of interest if you want to develop software for the STM32 microcontroller on the board. They enable a software switchable pull-up resistor for the D+ data line of the USB. This is especially interesting for the debugging of the STM32 software, so that the host is notified in case of a microcontroller reset that the USB device has to be enumerated again.

Unfortunately, the standard bootloader integrated in every STM32F302 does not support a switchable pull-up resistor, which is why there is a fixed pull-up resistor (R26) that takes over this function. Also interesting for debugging is the pin header P2, to which the SWD debugging interface was routed.

The PiDiver Development

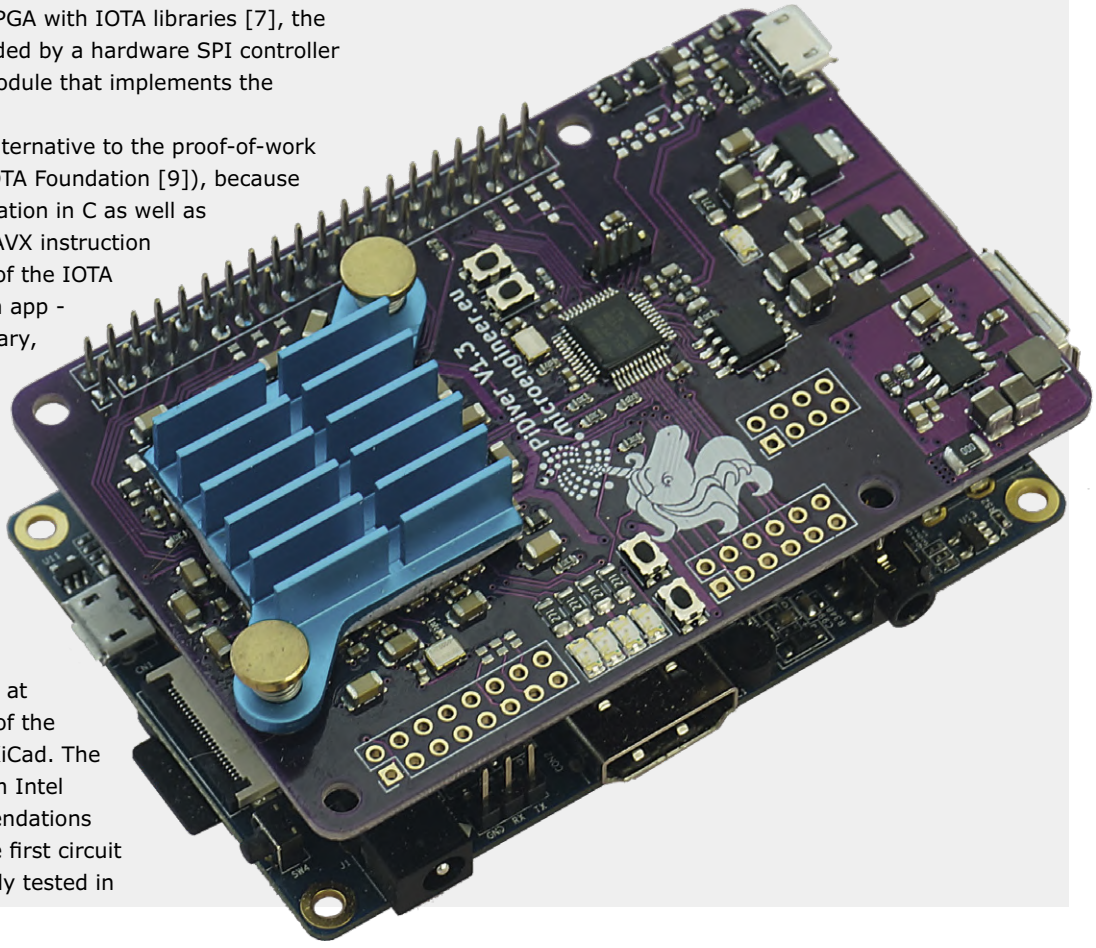
IOTA's proof-of-work is intended to protect the tangle from spam, but the task is also an obstacle for many valid applications that need to send data quickly to the tangle. The whole thing is very CPU-intensive even on full-grown computers and even more so for smaller IoT systems. For example, a Raspberry Pi takes 90 seconds on average to solve the PoW problem.

For this reason, the PiDiver project was launched. First, a proof-of-concept was developed on a ten-year-old development board (Altera DE1). The Altera Quartus 13.0.1 development environment was used because Cyclone II FPGAs are no longer supported by the newer versions. The PiDiver core was written in VHDL, based on the reference code "Pearl-Diver" (in Java) by Sergey Ivancheglo [12]. First a hashing performance of about 4.6 MHash/s was achieved, which could be improved to about 12.9 MHash/s a short time later. To be able to use the FPGA with IOTA libraries [7], the library "dcurl" [8] was extended by a hardware SPI controller for the Raspberry Pi and a module that implements the command set for the FPGA.

The library dcurl is a faster alternative to the proof-of-work library ccurl (library of the IOTA Foundation [9]), because it supports a PoW implementation in C as well as GPUs and CPUs with SSE or AVX instruction sets. The then official wallet of the IOTA Foundation [10] - an electron app - used ccurl as an external library, since the wallet itself was written in Javascript, which is relatively slow for PoW. This library could be replaced by dcurl. Later, a separate PiDiver library and a test program were developed in GoLang, so that dcurl was no longer needed.

During the first optimizations at the PoC the first PCB design of the PiDiver was developed with KiCad. The basic circuitry was taken from Intel data sheets, design recommendations were used for the layout. The first circuit board version was successfully tested in May 2018 with a hashing performance of approximately 14.8 MHash/s. The first version of the circuit board was tested in May 2018. In total there are four versions of the board. The first two versions were equipped with an STM32F1 microcontroller, which was replaced by an STM32F3 for bootloader functionality from the third version onwards. In the second version, an optimized power supply was used because the switching regulators on the first version were oversized. Version 3 has ESD protection and EMC filters on the USB, the two MAX40200 ("ideal diodes") and the DFU bootloader functionality have been added. The fourth version offers mounting holes for a heat sink as well as more compact (and therefore more) buttons and pin headers for your own extensions.

The firmware for the STM32 microcontroller was developed in C++ with Eclipse.



Most of the other components don't need any further comment (except that some of them are quite small), but special attention should be paid to the metal surfaces on the underside of the FPGA (U1) and the switching regulator (U3), which also have to be soldered. For manual assembly, therefore, large vias were placed under these components to allow soldering of these metal surfaces. It is recommended to heat the metal plates with hot air from the rear through the vias until solder melts on them. The use of a soldering iron is strongly discouraged here, as the heat supplied disappears very quickly in the intermediate layers of the board and the metal surfaces will prove difficult to heat to the required temperature. In the worst case even the circuit board will be damaged!

First commissioning

Once the circuit board has been assembled, it is advisable to check the voltages first. First, check the 1.2-V voltage at the output of the switching regulator. The core of the FPGA is (later) supplied with this voltage. It can quickly happen that the resistors of the feedback voltage divider are swapped, and the excessive voltage destroys the FPGA. If the voltage is okay, R8 can be soldered in.

Then you can check all voltages on the solder pads of the unassembled pin header J2 (**Figure 6**). However, the 1.2-V supply can only be measured there after the R8 has been assembled. If the board is supplied with power, LED D1 should light up and thus signal that the 3.3-V operating voltage is present. Afterwards it can be tested whether the STM32 microcontroller is recognized at the USB. To do this, connect a USB cable from the PC to the board while the BOOT button is pressed, whereupon the microcontroller logs on to the computer as "DFU Bootloader".

Software

Flashing the STM32 microcontroller

To flash the microcontroller, it must first be started with the DFU boot loader. To do this, keep the BOOT button pressed while connecting the board to the USB. Shortly afterwards the controller should log on to the computer as "DFU Bootloader". Under Linux you can use the tool dfu-util [3] to flash the firmware:

```
dfu-util -v -d 0483:df11 -a 0 -s 0x08000000 -D
   usbdiver.bin
```

After successful flashing and a reset, the controller should report as USBDiver.

GoLang Installation

The libraries were written in the programming language GoLang, which has to be downloaded and installed first. GoLang is available on [4] for multiple operating systems and architectures. You can also find installation instructions on the website. After installing and setting up the environment variables, restart the computer.

Installation of the test program

If GoLang has been successfully installed, the test program can be compiled. First you get (on the Raspi or the Linux-PC) the Git package:

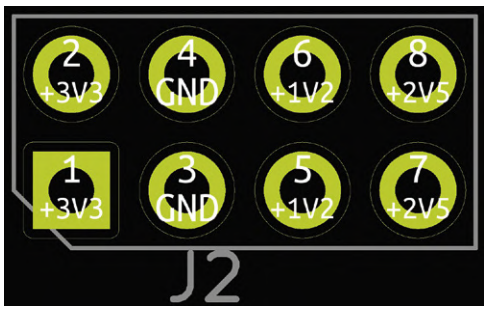


Figure 5: Assignment of the (unassembled) connector J2 for checking the supply voltages.

```
go get gitlab.com/microengineer18/pidiver1.3/golang/
   pidiver
cd $GOPATH/src/gitlab.com/microengineer18/pidiver1.3/
   golang/pidiver/main
go get -d ./...
go build
```

Now the actual test can begin. If you want to use the PiDiver over a Raspberry Pi, you start the program there:

```
$ sudo ./main -t pidiver
```

(Note: sudo is important, otherwise no direct hardware access to SPI is possible).
If you want to use the PiDiver via USB, enter the following on the Linux PC:

```
$ ./main -t usbdiver
```

The output looks similar in both cases:

```
...
2018/07/14 09:09:02 Found nonce: 0005e423 (mask:
   00000010)
2018/07/14 09:09:02 PoW-Time: 172ms (15.62MH/s)
2018/07/14 09:09:02 Nonce-Trytes:
   PIDIVER9V9999999OMKVVNGMMMM
2018/07/14 09:09:02 hash: RIRQYZKVJIDQIO9RBPLCJTSRTJ
   CWFVOODAMKXFHRJMFZQLR9AJPLCNYEPW9IJCEJWRBTVTRIWZY
   DA9999

2018/07/14 09:09:03 Found nonce: 00094b5e (mask:
   00000020)
2018/07/14 09:09:03 PoW-Time: 265ms (16.03MH/s)
2018/07/14 09:09:03 Nonce-Trytes:
   PIDIVER9T9999999UNGEKTVMMMM
2018/07/14 09:09:03 hash: EIW9MEACXYGVAQFCUGAKGMXKNC
   PGMMSPQWOLGMHZWHZKUEOLNRFKJGMNSELCSLRAEVCLYCBREBW
   TZ9999
...
```

The test program calculates noncents of random transactions and outputs the transaction hash, which in the case of the

IOTA main net always ends with “9999” (3 trits, which are 0, correspond to a 9 when output as a tryte).
There is also a server (running on the Raspberry Pi, but also on a Linux PC) that accepts the attachToTangle [5] command of the IOTA API and calculates noncents for real transactions. At this point, however, reference is only made to further documentation [6], as this would go beyond the scope of the article.

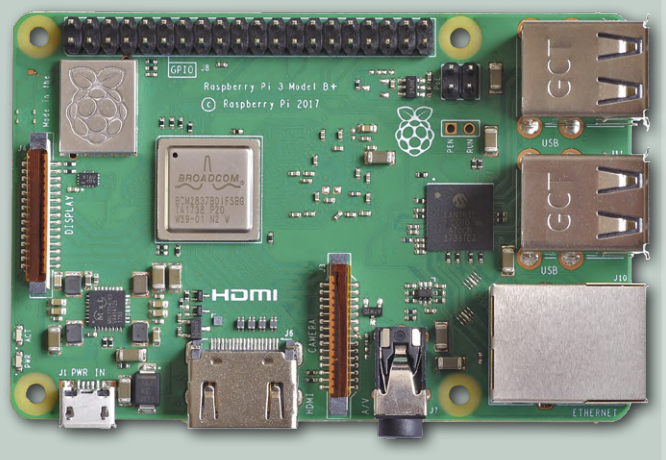
Power to the Pi!

For operation, it is important to use cables that can carry the necessary power. To power the Raspberry Pis you should use a USB cable as used for Powerbanks. The power supply must be able to supply real 5 V at real 2 A! Operation with the PC’s USB also requires cables that can handle at least 500 mA of current. If you use a wrong cable or power supply that is slightly weak on the chest, the FPGA usually loses configuration as soon as it starts proof-of-work. ⏮

180361-B-03

@ WWW.ELEKTOR.COM

→ Raspberry Pi 3 Modell B+
www.elektor.de/rpi-3b-plus



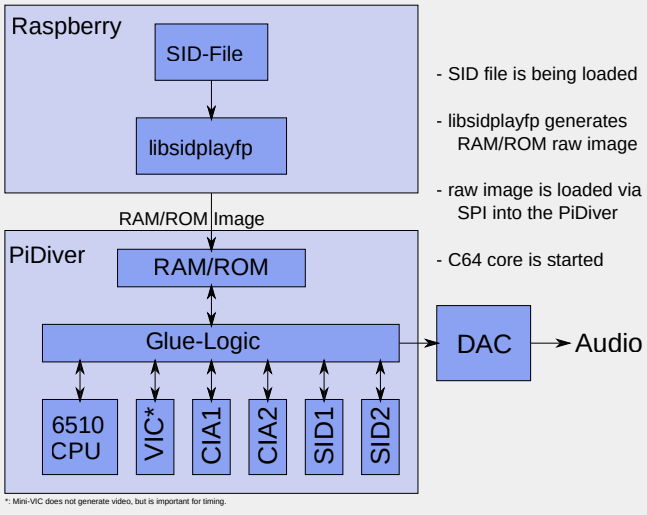
Web Links

[1] First part of the series: www.elektormagazine.com/180361-02
[2] GIT Project Repository: www.gitlab.com/microengineer18/pidiver1.3
[3] DFU Tool: http://dfu-util.sourceforge.net/
[4] GoLang: https://golang.org/
[5] Attach to Tangle: https://iota.readme.io/reference#attachtotangle
[6] PiDiver Documentation: https://ecosystem.iota.org/tutorials/pidiver-usbdiver-documentation
[7] IOTA Ledger: https://github.com/iotaledger
[8] Library dcurl : https://github.com/DLTcollab/dcurl
[9] Library ccurl : https://github.com/iotaledger/ccurl
[10] IOTA Foundation Wallet: https://github.com/iotaledger/wallet
[11] Musical PiDiver: www.youtube.com/watch?v=GhgDCf9oBEo
[12] PearlDiver: https://github.com/Come-from-Beyond/PearlDiver

Musical PiDiver

The PiDiver is basically a versatile FPGA board that provides unused pins for own applications via additional pin headers. For example, a C64 hardware emulation was tested that can play 8-bit 80s music in stereo via an I2S digital-to-analog converter connected to the board.

The library *libsidplayfp was used*, which provides a C64 software emulation that can be used to play SID music on a PC. The library internally generates RAM and ROM images that are executed on the emulated C64. Although the images were not intended to run on C64 hardware, it works fine. A big compliment to the developers of this library! A video with the music playing PiDiver can be viewed under [11].



The Elektor Community

82	203637	1040	546	235332
Countries	Enthusiastic Members	Experts & Authors	Publications	Monthly Visitors

Elektor breaks the constraints of a magazine. It's a community of active e-engineers — from novices to professionals — eager to learn, make, design, and share surprising electronics.

Elektor Web Store: 24/7 candy store for every electronics engineer! Permanent 10% discount for GREEN and GOLD Members.
www.elektor.com

Elektor Magazine: Six times per year a thick publication packed with electronics projects, news, reviews, tips and tricks.
www.elektormagazine.com

Elektor PCB Service: Order your own PCBs, both one-offs and larger runs.
www.elektorpcbservice.com

Elektor Weekly & Paperless: Your digital weekly news update. Free.
www.elektor.com/newsletter

Elektor Academy: Webinars, Seminars, Presentations, Workshops and DVDs ... Practice-oriented learning.
www.elektor-academy.com

Elektor Books: Arduino, Raspberry Pi, ESP32, IoT, Linux and more. Available in our online store with a 10% Member discount!
www.elektor.com/books

Elektor TV: Reviews, timelapse, unboxing and personal journals. Watching is learning.
www.elektor.tv

Elektor Labs: Showcasing your own projects and learning from others. We develop and test your ideas!
www.elektormagazine.com/labs

Become a member today!

GREEN
€5.67 per month
£4.08 / US $6.25

- ✘ 6x Elektor Magazine (Print)
- ✔ 6x Elektor Magazine (PDF)
- ✔ Access to Elektor Archive (Thousands of Articles)
- ✔ Access to over 1,000 Gerber files
- ✘ Elektor Annual DVD
- ✔ 10% Discount in Elektor Store
- ✔ Exclusive Offers

www.elektor.com/green

GOLD
€7.58 per month
£5.50 / US $8.42

- ✔ 6x Elektor Magazine (Print)
- ✔ 6x Elektor Magazine (PDF)
- ✔ Access to Elektor Archive (Thousands of Articles)
- ✔ Access to over 1,000 Gerber files
- ✔ Elektor Annual DVD
- ✔ 10% Discount in Elektor Store
- ✔ Exclusive Offers

www.elektor.com/gold

FREE

- ✘ 6x Elektor Magazine (Print)
- ✘ 6x Elektor Magazine (PDF)
- ✘ Access to Elektor Archive
- ✘ Access to over 1,000 Gerber files
- ✘ Elektor Annual DVD
- ✘ 10% Discount in Elektor Store
- ✔ Elektor weekly e-zine
- ✔ Exclusive Offers

www.elektor.com/newsletter

facebook.com/ElektorLabs

twitter.com/Elektor

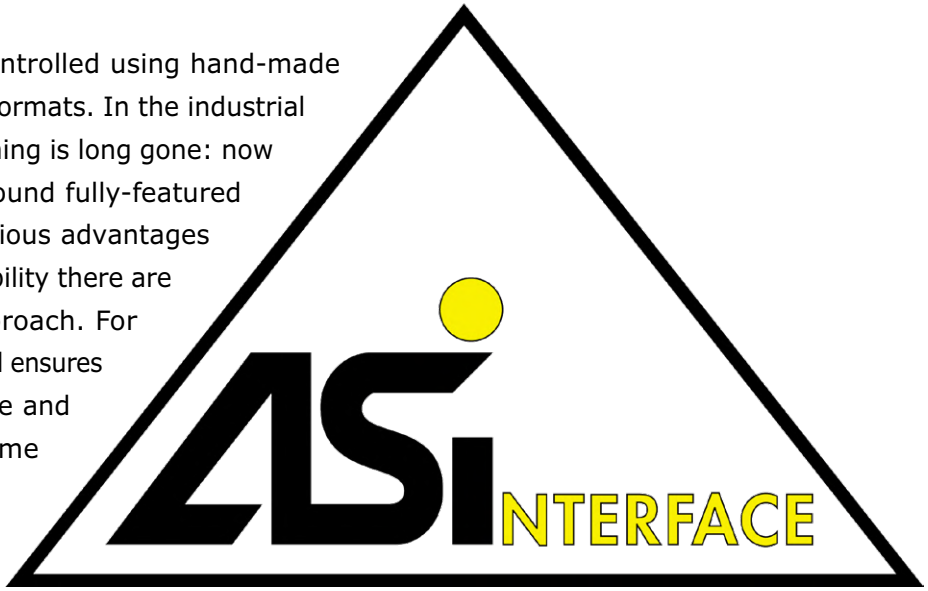
instagram.com/elektorlabs

AS-Interface Essentials

How the industrial automation bus system works

By **Tam Hanna** (Slovenia)

In the olden days everything was controlled using hand-made custom cabling and proprietary data formats. In the industrial world at least, however, this kind of thing is long gone: now entire systems are based instead around fully-featured bus architectures. As well as the obvious advantages of easier maintainability and expandability there are other reasons to prefer such an approach. For example, a widespread industry standard ensures the long-term availability of hardware and software components, at least to some degree. In this article we will take a look at AS-Interface.



The approach to automating more complex manufacturing processes can be thought of as a series of layers as illustrated in **Figure 1**. At the bottom level we find the sensors and actuators. For reasons of cost and efficiency, communicating with these devices requires us to go to the trouble of using Ethernet-style cabling: short-range connections using protocols such as SPI or I^2C are often no match for the harsh environment of an industrial installation [1].

Two widespread techniques for connecting sensors and other devices are AS-Interface (actuator-sensor interface, also known as ASi) and the IO-Link protocol, standardized under the name SDCI, and promoted for use in PROFIBUS environments. The most significant advantage of AS-Interface is that a master can talk to many slaves using a single port. IO-Link, in contrast, normally requires a separate port and cable for each slave. In the English-speaking world at least, IO-Link is generally considered more suitable for use in systems where large quantities of data must be communicated.
Note that ASi and IO-Link can to some extent coexist. For example, the system from Bihl+Wiedemann shown in **Figure 2** and described in detail in [2] uses IO-Link to collect information; the compiled datagrams are then passed to the master using ASi.

Which version?

AS-Interface came into existence in 1990 from the formation of a group of companies under the name 'AS-International Association' [3], a kind of special interest group. This group holds the rights to the name 'AS-Interface' and sells products based on the official standard. In case you might be considering joining the group, you may wish to bear in mind that membership will set you back in the region of €4000 per annum.
If you would like just to take a look at the standard there is a more affordable path available: AS-Interface is specified via a bundle of industrial standards, of which by far the most popular is EN 62026-2:2015. This standards document can be obtained through the official channels for a mere €250. Alternatively, if you fancy a spot of personal research, it should be possible to find the document in the library of a local technical college or university; and failing that, perhaps an inter-library loan or a suitable institutional library might come to your assistance.
One possible complicating factor in getting to grips with all these materials is that currently there are two versions of ASi on the market. Of these, only version 3.0 has a significant level of availability, both in terms of specifications and in terms of hardware. However, version 5.0 has been widely promoted since the 'SPS IPC Drives 2018' trade show: the new version improves on the old in several details.
Although such developments are perfectly normal in themselves, a sticking point is that the exact specification for version 5.0 of the protocol is, at the time of writing, not available to all members of the association. According to insider information the information is only available to those directly involved in developing the protocol.

An interesting result arises from this. Transceiver ICs for version 3.0 (on which more below) originated at the company ZMD, which was then bought out by IDT, which was in turn swallowed up by Renesas. Hence Renesas is the partner working on support for version 5.0, and so they can proclaim 'we implement all the protocols'.

The ASi cable

A fundamental aspect of the implementation of the 'multi-drop' capability of ASi lies in the construction of its cable. There are two conductors in the cable, which carry the signals AS+ and AS−. Here AS− is the 'ground' connection relative to which AS+ carries the positive data signal. But note that it is under no circumstances permissible to connect the AS− signal to earth.

In actual installations the ASi cables are almost invariably yellow in colour. The asymmetrical shape shown in **Figure 3** is typical, the form of the cross-section being reminiscent of a nose. An optional black cable with the same construction can be employed to deliver additional power to devices on the bus. Most ASi devices have connectors to fit a cable of the type shown in Figure 3, equipped with teeth that pierce the insulation of the cable when the connector is fitted in order to make electrical connection with the two conductors inside. Adding a new device to an ASi bus is therefore reasonably easy and 'failsafe'. Although this design is common and practical, it is possible to use the ASi bus with other types of cable.

When sensors are connected directly to microcontroller systems using the 1-wire bus it is possible for low-power devices to take their supply directly from the bus signal. The same goes for AS-Interface devices: the nominal bus voltage is 24 V, and according to the specification the bus can in theory carry up to 8 A. In practice, however, it is recommended to maintain the current draw below 2 A in order to keep voltage drops in the cabling and connectors within reasonable limits.

An important point is that the AS-Interface signals are not at a fixed potential with respect to earth. Indeed, as mentioned above, it is explicitly forbidden to earth the cable. As well as fully-networked ASi slave devices it is also possible to have devices that use the bus only as a power source. They can draw up to 400 mA at 24 V, but are not allocated an address and do not take part in communication. Since an ASi bus can be up to 100 m long (and, with suitable termination, even up to 300 m long), it is also a convenient way to provide a supply to 'alien' systems with modest power requirements. Such devices are simply connected to AS+ and AS− and get on with whatever their business might be.

An ASi system is normally powered from a dedicated power supply, whose construction is illustrated in **Figure 4**. Designing and building a supply of this type yourself for a professional application is strongly discouraged: on the website of the ASi association [3] there are links to many companies who will be happy to sell you suitably certified products.

It is interesting to note that the power supply is expected to provide a DC output at between 29.5 V and 31.6 V. This level, rather more than 24 V, is needed to compensate for voltage drops in cables and connectors. Also, the high level of noise immunity offered by the ASi system is not carte blanche for sloppy or disorganized cabling. When an ASi cable is run in the same duct as mains cables carrying power-line signalling, it is possible in some circumstances for the network to suffer inter-

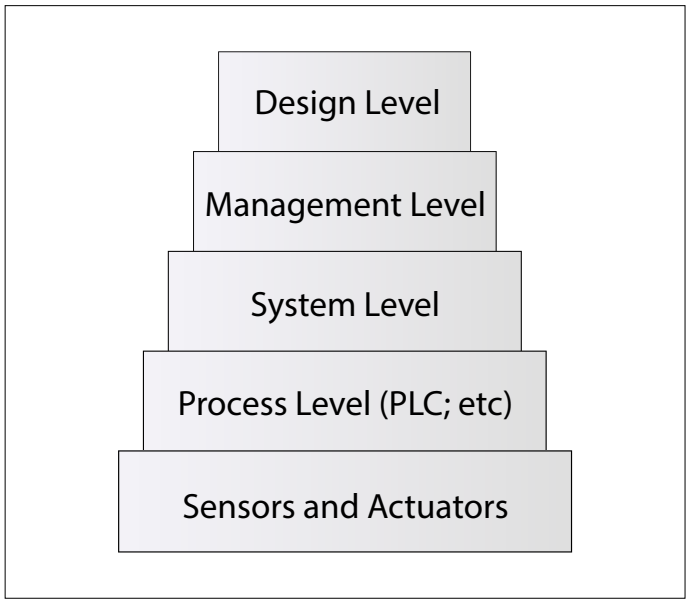


Figure 1: The hierarchical levels of an industrial automation system.

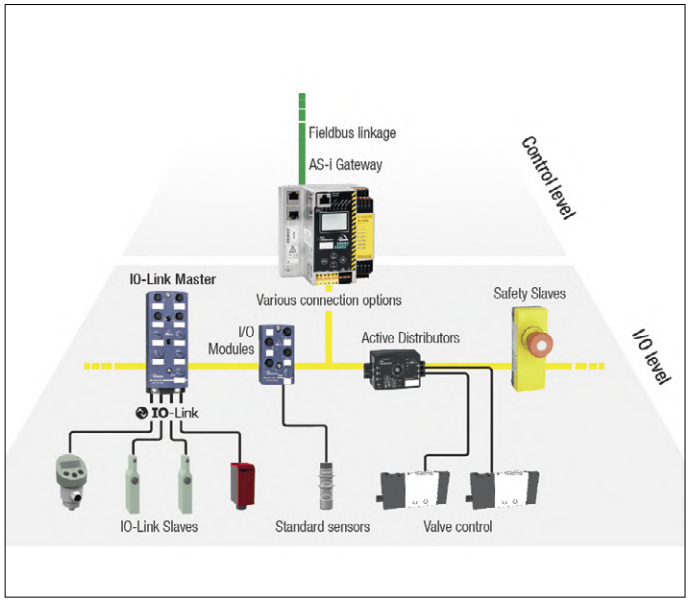


Figure 2: From Fieldbus to sensors and actuators (source: [2]).

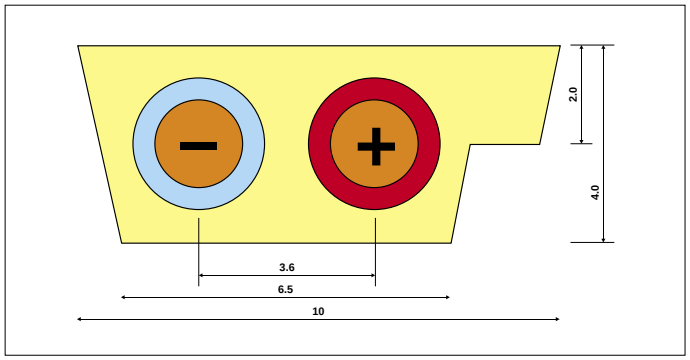


Figure 3: Cross-section of an ASi cable (conductor diameter 1.8 mm). The 'nose' shape prevents accidental reverse-polarity connection. (Image: after Bihl+Wiedemann).

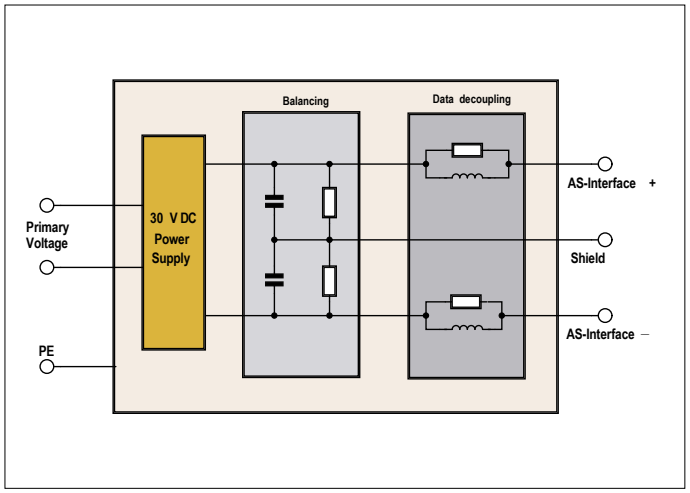


Figure 4: The PELV power supply has a comparatively complex internal construction (source: [4]).

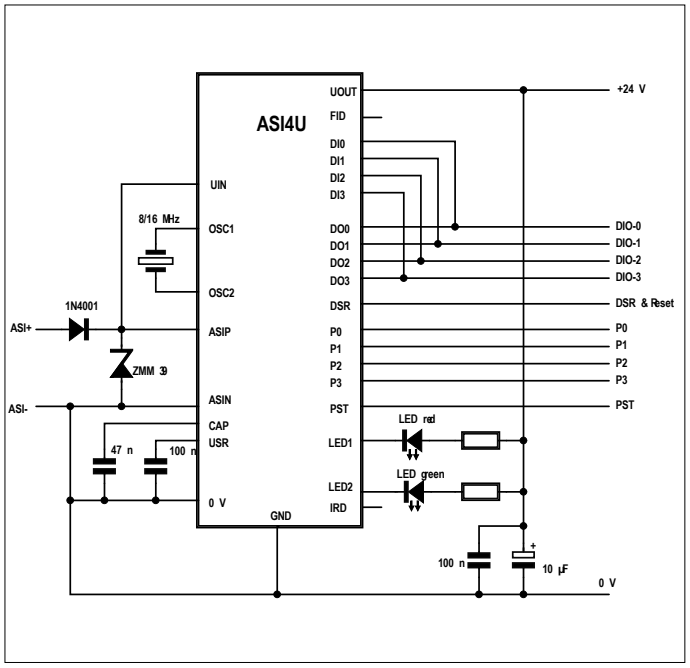


Figure 5: The transceiver chip requires very few external components (source: [6]).

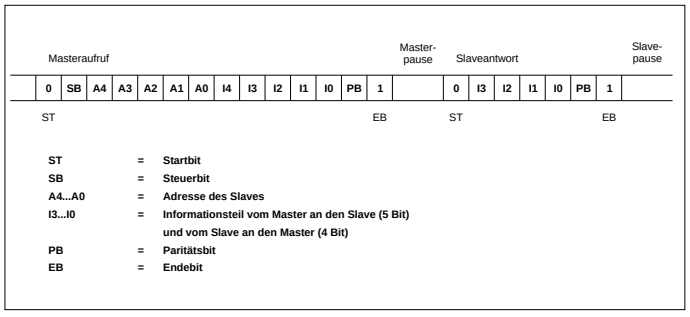


Figure 6: Data format of an ASi message (source: [7]).

ference that is hard to get rid of. A presentation by Phoenix Contact [4] gives further advice on how to arrange an installation.

Communication

It is clear from the cable construction shown in Figure 3 that there are no separate connections for power and data. It follows that before a receiver can process a message the signal must be transformed or demodulated.

Data are sent using APM (alternating pulse modulation). This is a scheme based on Manchester coding, where the data clock and the bitstream to be sent are combined. Then the Manchester-coded signal is used to modulate a current to transmit the data. According to the freely-available information the amplitude of this modulation is between zero and 60 mA.

Negative-going edges are represented by a rise in the transmitted current and positive-going edges by a fall. Because of the inductance of the cable these changes in current are converted into voltage impulses, with excursions of around ±2V around the nominal supply voltage. It is important to note that the beginning of a datagram is always marked by a negative pulse: this is relevant when it comes to triggering digital oscilloscopes and signal analysers.

Unfortunately the usual makers of oscilloscopes have so far shown little interest in incorporating ready-to-use ASi protocol decoders into their products. Only the range of devices from Pico Technology support the protocol, although even then not natively: a third-party tool uses the API to provide analysis functions [5].

If you wish to build an ASi device it is not really practical to implement the decoder hardware yourself. The company IDT mentioned above offers the ASI4U transceiver chip, which can be configured to operate as a master or as a slave. **Figure 5** shows the basic circuit, taken from the data sheet.

Particularly noteworthy are the two light-emitting diodes, which IDT's data sheet labels as green and red. The ASi bus specification requires that these LEDs are present on slave devices, and they give information on the current operational state. More information on this can be found in the device data sheet. The pins labelled DIx and DOx are input and outputs. If a module using the transceiver is operating in slave mode, then this is where data communicated over the bus will be made available. The pins labelled Px are used to configure parameters, and are particularly important in the master mode, which we will not go into here.

The communications standard

Now that we have understood the underlying electrical properties of an ASi system, we can turn to the logic behind the communication format. All communications are strictly under the control of the master, which carries out a cyclic polling process whereby it interrogates each individual slave connected to the bus in turn. Each slave will normally reply to this with a four-bit payload. When a cycle is complete it starts again from the beginning: see **Figure 6**.

The format of the address determines the maximum number of devices on the bus. In 'normal' datagrams there are just five address bits available, and address zero is reserved for a new slave. There are thus 31 available values remaining for addressable slaves. A new slave device, which initially has address zero, is allocated a new address from the available pool by the master.

@ WWW.ELEKTOR.COM

→ MonoDAQ-U-X (50 kSa/s) multifunctional USB data acquisition system
www.elektor.com/monodaq-u-x-multifunctional-usb-data-acquisition-system-50-ks-s

→ PicoScope 2205A
www.elektor.com/picoscope-2205a



A consequence of the cyclic polling is that the response time of an ASi system is deterministic. In ASi 3.0 the response time is normally around 5 ms, while the newer version 5.0 with 24 bus participants promises a cycle time of just 1.2 ms. Also, the maximum number of slaves in ASi 5.0 is considerably greater, at 96.
It is particularly important to realize that either data or parameters can be transmitted. One of the information bits indicates which type of information is being transmitted on the bus. The control bit specifies whether an address or a command follows.

Where do we go from here?

If you are to work on an existing ASi system, or if you would like to (or have to) build a new ASi system, it is important to think rationally. Developing your own modules and sensors may sound like fun, but in practice it is not economic. There are many companies working in this field, and they offer practically any sensor and actuator type you can think of with AS-Interface capability. It is much more efficient to assemble your complete system from these components and sell it, along with consultancy support services, to customers.
If you are not put off by the above and still wish to experiment with ASi, it will not really be possible to avoid taking up membership of the standardization organization sooner or later. The standard includes all the necessary information, and according to oemsecrets.com the ASI4U transceiver chip is available on the open market, but there is a big 'however'.
The problem relates to intellectual property rights. If, without explicit permission, you use the term 'ASi' or the ASi logo, you will rapidly find yourself in legal hot water. And if you attempt to sell your system on the open market, it is only a matter of time before someone will come knocking on your door.
This situation is not exactly satisfactory for hobbyists interested in experimenting in the lab. On the other hand, working with ASi systems can prove very lucrative in the longer term, as salaries for developers and consultants in industrial automation are in general comparatively high: that might be some compensation for all the formalities and travails. ⏮

190124-02

Webinar

A brief overview of the basics of the ASi standard can be found at Pepperl+Fuchs' YouTube channel. Start here:

www.youtube.com/watch?v=ytX5JOh2_bg

Reading the data sheet is not optional!

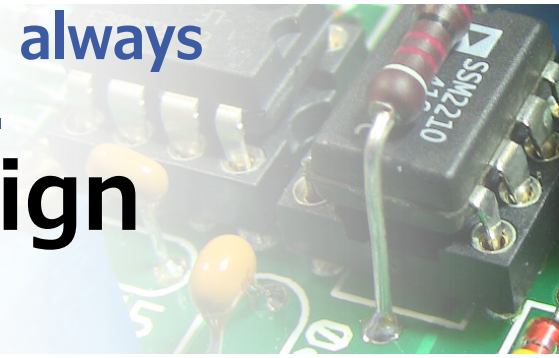
For reasons of space this article is only able to give a broad overview of the ASi standard. Before taking any practical steps we therefore strongly suggest that you make a careful study the data sheet of the IDT ASI4U device [6].

Links and literature

[1] J. M. Jacob, Industrial Control Electronics, page 260 ff.
[2] ASi information: https://www.bihl-wiedemann.de/en/applications/communication/as-interface-and-io-link.htm
[3] AS-International Association: https://www.as-interface.net/en
[4] Phoenix Contact: www.phoenixcontact.com/assets/downloads_ed/global/web_dwl_technical_info/instrecomm10_e.pdf
[5] Pico Technology ASi tool: www.picotech.com/library/picoapp/as-iexpert-network-diagnosis-tool
[6] ASi data sheet from IDT: www.idt.com/document/dst/asi4u-datasheet
[7] DESY (in German): http://www-mks2.desy.de/content/e3740/e5177/e7190/e7997/e8006/e9274/e9656/index_ger.html

(Almost) everything you always wanted to know about...

Analogue Design and Circuitry



With answers from **Ton Giesberts** (Elektor Labs)

Q How can you keep noise in an amplifier as low as possible?

A Choose resistor values that are as low as possible, but make sure that the currents are not higher than the amplifier or the individual stages can supply. The feedback circuit is also a load for the amplifier. Keep an eye on the maximum current needed by the load, and maintain a good safety margin. Too much current will also cause more noise and increase the distortion. Avoid oscillations. HF oscillation causes a lot of noise, and it can disturb DC settings and cause higher distortion. If you have opamp circuits with unity gain, make sure you use an opamp type that is unity gain stable. Some opamps are only stable with an amplification factor of 5 or 10. Another trick that's often used with phono preamplifiers is to connect transistors in parallel in the input stage.

Q How can you keep the distortion in a load as small as possible?

A Connect the grounds of the input signal, the feedback and the load to the same point, which is called the star point. The power ground connection must be tapped off from this point. Avoid any other branches in the three connections.

Q How can you keep the DC offset in an amplifier as small as possible?

A The simplest solution is to use decoupling capacitors at the input and output. However, that is not possible if the amplifier has to pass very low frequencies or DC. For very low frequencies the capacitors would be much too large. A better solution is to ensure that the DC currents at the amplifier input are as small as possible. With bipolar opamps the input currents are often in the microampere range. So it is better to use JFET types, which have input currents in the low nanoampere range. The best option is to use types with MOSFET inputs. However, JFET types and in particular MOSFET types have higher input noise, especially at low frequencies. Which of the three types you use also depends on the dimensioning and frequency range of the circuit (see Question 1). Always carefully check the information and specifications in the manufacturer's data sheet.

Q How can you protect an amplifier input against overvoltage, and what do you need to pay attention to in this regard?

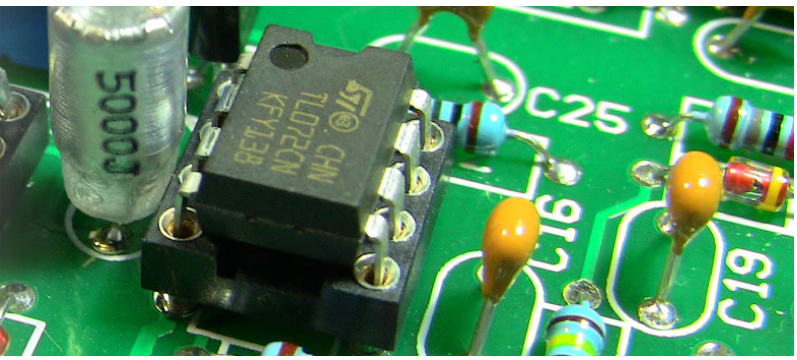
A A resistor in series with the input is one option, but that increases the noise level and susceptibility to interference. It can also degrade the bandwidth. Using clamping diodes tied to the supply voltage rails is a better option. Then you can use a much lower series resistance. However, the diode leakage current can cause a large offset with a very high input impedance, such as with an active probe. In that case you must use special diodes. The diodes will only affect the bandwidth with HF circuits. You should bear in mind that the supply voltage can be lifted if the current through the clamping diodes is too large. To prevent that, the regulator for the supply voltage must be suitably designed and protected. Another simple solution is to connect a pair of Zener diodes in reverse series configuration in parallel with the input (or a number of diodes in series connected in anti-parallel). However, they may influence the input impedance and the linearity to a degree.

Q Why is there an analogue ground?

A In a circuit with analogue and digital parts, a separate ground is often used for the analogue part. It is clearly separated on the PCB to avoid disturbances from the digital circuitry. In many cases there is also a separate supply voltage for the analogue circuitry. In that case it is a good idea to keep the supply currents and the different parts of the circuit well separated from each other. With very sensitive circuits, it may even be necessary to provide a separate ground for the input portion of the analogue circuitry.

Q How can you avoid disturbances in and from analogue circuits?

A Proper decoupling of the supply voltage is a good start. One example is ferrite beads in series with the supply lines of switching circuits. Depending on the bandwidth of the disturbance signal, using SMD components may be the only solution, with increasingly smaller components necessary at higher frequencies. They must be placed as close as possible to the disturbance source. Also pay attention to common-mode suppression, since most of the interference generated by circuits of this sort is specifically common-mode. In practice, this means that the circuit, and in particular connecting leads, act as an antenna and can therefore pick up interference. That's why you see ferrite beads on mains power cables, computer cables and the like. The bandwidth and slew rate of the circuit should also not be greater than strictly necessary. Avoid wide loops in the PCB layout; the return current should flow as close as possible to the supply current. If this cannot be achieved by a trace or a copper surface next to or beneath the supply



trace, then designing a PCB with four (or more layers) is virtually unavoidable. Shielding is often necessary with HF circuitry. Here as well you should check if common mode suppression is necessary for the input and output signals.

Q What can cause hum in an amplifier?

A This question has already been partially answered. Here we aren't talking about loose or unconnected cable shields or unintentional swapping of the signal conductor and ground. The cause could be the magnetic field of a power transformer in the vicinity of the circuit. In many cases, this is dealt with by using a ring core (toroidal) transformer, but ring core transformers also have disturbance fields and the actual benefit is often less than expected. Shielding with mu-metal may be a solution. A simpler solution is a large distance between the transformer and the circuit, but that is often not possible for practical reasons. An advantage of E-core transformers is that the metal sides where the laminated core is clearly visible have a much smaller disturbance field than a ring-core transformer. For that reason, you sometimes see this type of transformer mounted at a 45 degree angle in a corner of the enclosure, so that these two worst disturbance fields are located outside the enclosure. Another source of disturbance is bridge rectifiers, which is why four capacitors with a value of 10 to 47 nF are often connected over the rectifier terminals. Particularly with sensitive circuits, such as microphone amplifiers, this disturbance is a potential source of hum. However, it can also cause problems with radio receivers and other HF circuits. In your PCB designs, make sure that capacitor charging currents do not flow through or near signal grounds. Currents through the neutral point must return to the transformer along the shortest possible path. Here as well, keep the distance to the most sensitive parts of the analogue circuitry as large as possible. Oscillation – in an amplifier, for example – can also cause hum. If that happens, the currents can be much higher than usual, causing the supply ripple voltage to rise considerably.

Q What is the best way to convey small signals?

A The simplest way to convey a signal is to use a signal line with an associated ground. This is called 'asymmetric' or 'unbalanced'. In the case of a cable, a coax type is often the preferred choice, but a pair of twisted conductors may also be sufficient. However, there may be potential differences between the grounds of two different equipment items, causing currents to flow though the ground connection. A cable in a disturbance field can also cause this sort of noise. These problems can also occur with connections between two parts of a circuit on a PCB. A better option is to use a symmetrical signal, particular when transmission over a relatively large distance is required. That is also called a balanced signal. This actually the only option for achieving a greater dynamic range, which means how well the circuit is able to process very small signals. With a balanced signal, you have two identical signals with opposite phases. Noise on both signal lines will be suppressed by subtraction from the two signals. The resulting useful signal is twice as large as each individual signal of the balance connection. This is why high-end A/D converters with relatively high resolution usually have balanced inputs.

Q What is often overlooked when making measurements on a circuit?

A When you use a multimeter, you should take the internal resistance into account for both voltage and current measurements. In both cases, a measurement error can occur, and with sensitive circuits, touching a probe to the circuit can also cause instability or even oscillation. That often results in a much larger measurement error. An disturbance signal can also be injected into the circuit, which is something you should bear in mind. In case of doubt, you should use an oscilloscope to view the signal. Oscilloscope probes are also not perfect; they have capacitance (and resistance) that can also cause other problems. An active probe can be helpful for very sensitive circuits and/or very high frequency signals; active probes often have a higher input impedance and lower capacitance.

Q What types of capacitors should be avoided in audio circuits?

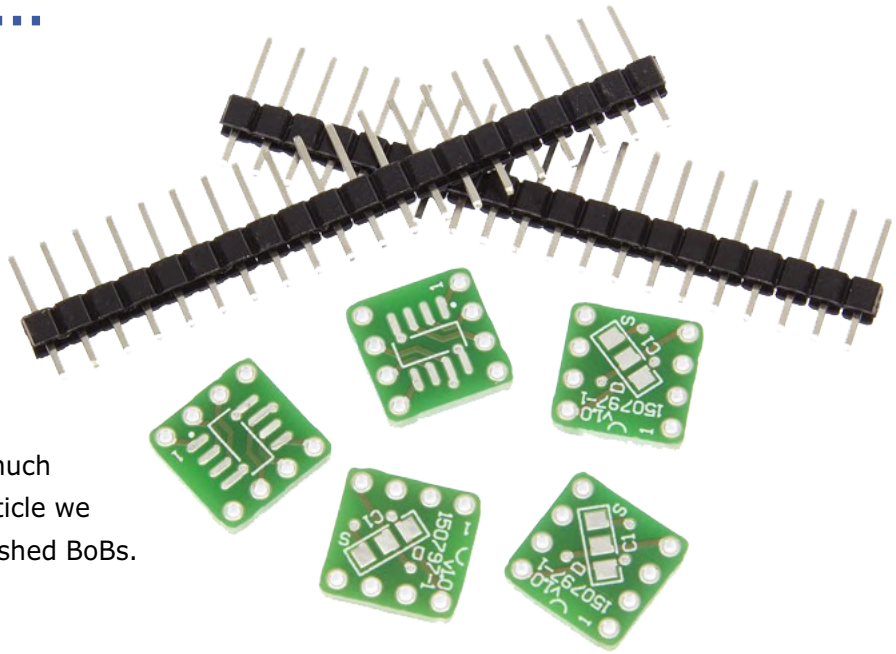
A omething that many people do not realise is that capacitors — depending on their construction and even more on their dielectric material — are far from ideal, and that they can considerably degrade the quality and performance of a design. Ceramic capacitors are certainly the worst. Depending on the dielectric material, the capacitance is voltage and temperature dependent, and they also suffer from microphonic effects, which means that mechanical vibration or stress can generate a voltage in the capacitor. In addition, they suffer from ageing. In the case of electrolytic capacitors, the capacitance is not stable, and in many cases the value is chosen much larger than necessary to avoid distortion at a corner frequency, with the result that the corner frequency is much lower than intended. If you are interested in the subject, try searching for the term 'dielectric absorption'. A known phenomenon is that after a capacitor has been discharged, the voltage on the capacitor terminals slowly rises. It is better to use capacitors with plastic films, such as polyester, polypropylene or polystyrene. The quality often increases with decreasing dielectric constant. A disadvantage with the better types, such as polystyrene, is that the capacity range is small, with a maximum of several tens of nanofarads in many cases. Consequently, the most suitable type is in part determined by the specified value and/or the size of the capacitor. ⏮

190099-03

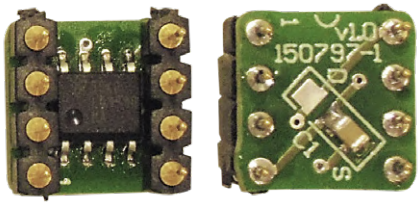
An Elektor BoBs Catalog

It's the little things…

Contributed by **Jelle Aarnoudse** (Netherlands)



At the Elektor Lab we regularly design small, but darn handy, Break-out-Boards (BoBs). Although these are frequently very popular we nevertheless often don't devote a complete article to them, because there is not always that much to say about them. That is why in this article we provide an overview of the recently published BoBs.



SO8-to-DIP8 Adapter (no. 150797)

We start with a simple and very small Bob, partly because it was the reason for this article. This small printed circuit board, about the size of one quarter of a postage stamp, can be used to enable modern, often better, chips that only come in an SMD package to be used in an older circuit board that was designed to accommodate a DIL8. The chip is mounted on the bottom between the header pins. A decoupling cap can be mounted on the top between pin 4, usually ground, and pin 7 (for opamps, for example) or pin 8 (many other ICs). Available in sets of five from the Elektor Store.

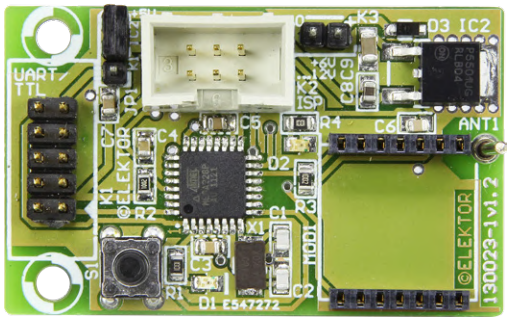


MEMS Microphone Adapter (no. 150346)

It can be even smaller! For the Bat Detector[PLUS] we required a MEMS-microphone which is so tiny and difficult to solder that we made a small adapter board for it. The module is available ready-made from our Elektor Store. Considering the excellent specifications for this microphone, this BoB is suitable for many other applications.

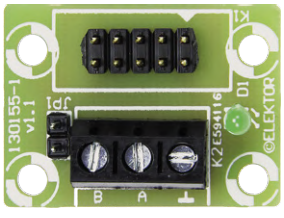
Wireless Gateway (no. 130023)

If something is going to characterize the developments in electronics for the next few years then it the trend of everything becoming wireless. And then not in the way that we have become accustomed in the HMI (Human-Machine Interface), but the MMI (Machine-Machine Interface), or IoT (Internet of Things). That is why there is an urgent need for simple and, at the same time, secure wireless connections. This is the reason why there has been, and continues to be, a rapid rate of development of chips and modules. But the datasheets of these are often a complete book and the application of which is not user-friendly. That is why we in the Elektor Lab have developed a number of BoBs with a 433MHz-transceiver, an ATmega328 and a UART. With the help of the pre-programmed firmware these boards are easily employed as a wireless gateway. Command and data strings enter your local aether via the UART. Again available from the Elektor Store as a ready-to-go module.

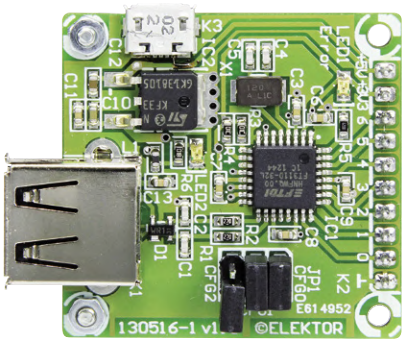


RS485 Module for ECC (no. 130155)

For less complicated communication tasks than the one described above, it is very handy if you can quickly connect something. For this purpose there is the RS485-module for ECC (Embedded Extension Connector). This can be used to provide a microcontroller board with an RS485-interface, which is suitable for reliable communications, also over larger distances and for bus configurations. Available ready-made from the Elektor Store.



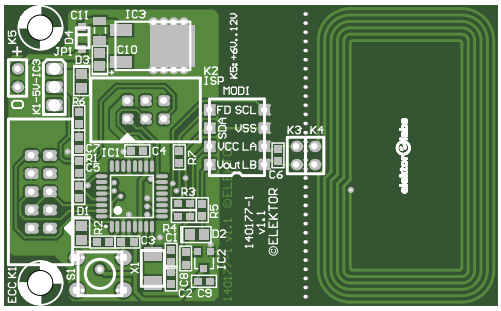
FT311D BoB (no. 130516)



For controlling and testing of boards you have built yourself you will nearly always require some input and output interface. Now nearly everyone has a smartphone, which is eminently suitable for this. The FT311D-chip has been especially developed to connect Android devices via USB to external electronics. Now simplicity is not an obvious characteristic of USB, but this chip has a USB-host interface and various interface pins for connecting your own electronics; with the FT311D that amounts to seven. In addition, the manufacturer of this chip has already loaded the firmware that provides a complete USB communications via the AOAM protocol. In the Elektor Store both a bare circuit board as well as a completely built module are available.

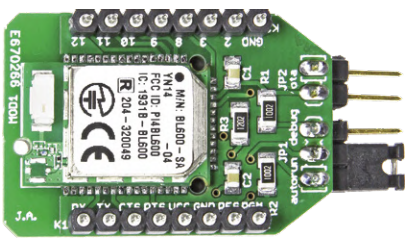
NFC Gateway (no. 140177)

This is again a communications BoB. Many phones these days are equipped with NFC, which can be used to exchange information across a distance of a few centimeters, for example for (small) payments. This, as electronics enthusiasts, we find obviously terribly interesting. How does it work; can we read such an 'NFC-tag' or even write it? Yes, this can all be done, but it is not all that easy. That is why we developed an NFC gateway for our readers, containing the well-known ATmega328, which takes care of the the protocol matters and in addition contains an NFC-tag and an integrated antenna. This NFC-tag is so difficult to solder that we supply it is a ready-made Bob, the gateway in contrast is available as a bare circuit board.



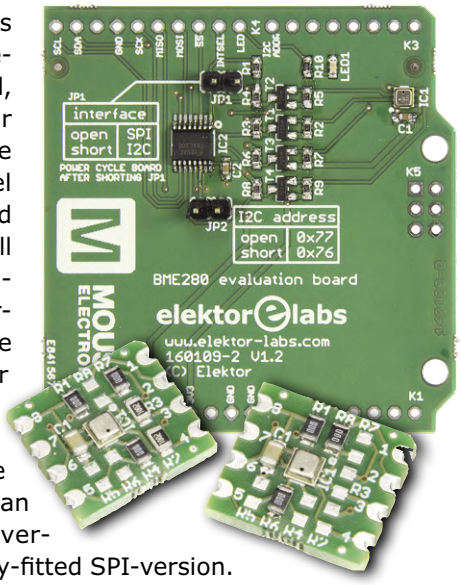
BL600 e-BoB Bluetooth Low Energy Module (no. 140270)

And another wireless module! The BL600 e-BoB is a breadboard-friendly interface board for ultra-frugal radio communications. Even for long-term use a button cell such as a CR2032 will suffice! So eminently suitable for IoT projects up to about ten meters, where an off-line power supply is not desirable. A ready-to-go module available from the Elektor Store.



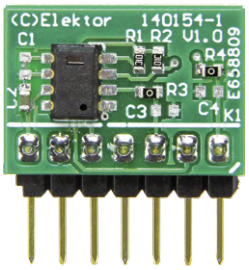
BME280 Arduino Shield (no. 160109)

The BME280 from Bosch is a sensor which can measure temperature, humidity and pressure. This can therefore be used to make a single-chip weather station. With this Arduino-shield developed by Elektor Labs, this chip can be comprehensively evaluated, using either SPI or I²C mode. Because of the built-in level shifter this can used in both a 3.3 as well as a 5V environments. Various versions are available from the Elektor Store: the bare circuit board, the bare version of the evaluation board, an already fitted I²C-version and an already-fitted SPI-version.



CC2A-eBoB Temperature/Humidity Sensor (no. 140154)

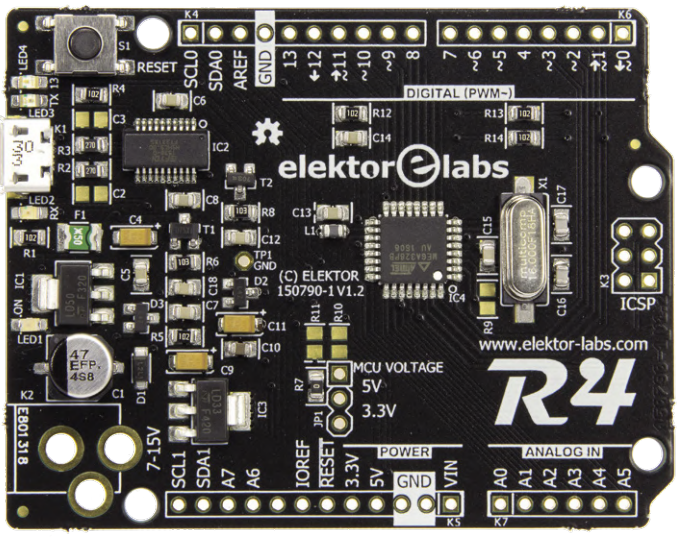
This fantastic sensor has an accuracy of 2% and are individually calibrated and tested. And although is comes in an SMD package it is nevertheless still quite easy to solder by hand. However: it doesn't fit in a breadboard of course. That is why we made this small circuit board with a standard pinheader.



Elektor Uno R4 (no. 150790)

And finally, this overview cannot forget to include our own Elektor Uno R4-board, a continued development of the now well-known Arduino R3 board. The R4 has more interface options, is more robust and also more versatile. The ultimate platform for your own IoT developments. ⏮

(150797)



Err-lectronics

Corrections, Feedback and Updates to published articles

A Monster LED Clock with Wi-Fi and Temperature Display



ElektorLabs 3/2019, p. 60 (180254)

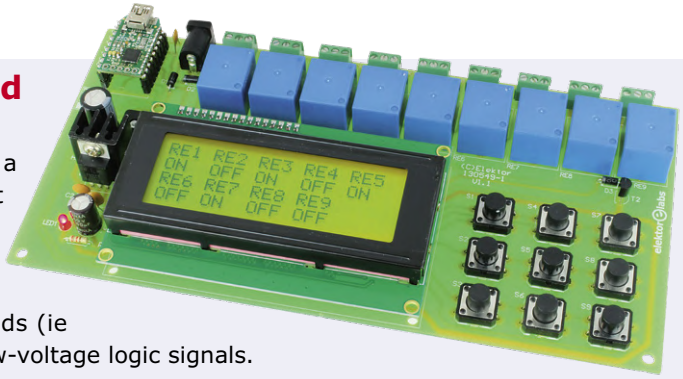
FEEDBACK. This article really sparked my curiosity. What was interesting was the connection of sensor boards with the clock using MQTT. In this way, the project can also be linked to other projects, e.g. with the IoT Gateway from the March & April 2017 edition. If you run an RPi, then you can also install FHEM and use the sensor data there as well. From the IoT gateway you could possibly develop a low-energy power-saving sensor for battery operation. The monster clock display could then show other information such as when mail has arrived. A dot matrix display could be implemented so that you could then display text messages. The clock could be a good platform for a whole series of projects. You have probably already thought of a lot more possible applications also.

Frank Klee

Thanks for the feedback. In fact, there are further modifications planned for the clock and we would like to incorporate the facility to exchange data via MQTT into several products. A link between the clock and the weather station is planned, as well as a modification that will let the clock signal when a doorbell is rung. What is also under consideration is to use the free IO pins to connect a light sensor or a DHT22 temperature and humidity sensor and show the readings on the display. These new features are of course updates to the original design and will be presented in an upcoming issue.

Mathias Claußen (Elektor Labs)

9-Channel Relay Switcher Board



ElektorLabs 3/2019, p. 80 (190027)

FEEDBACK. Greetings from Austria! Will there be a complete kit of parts for this project? It's exactly what I need. However, I have one reservation — the contacts of the relays used can handle 1200 W AC, but the maximum voltage is not specified. Although 230 V mains looks possible the creepage separation between the relay contact pads (ie the centre contact) are quite close to the tracks carrying low-voltage logic signals.

Manfred Tischler

Yes this project will soon be available as a kit of parts (we are currently awaiting its arrival).
The board is designed to switch 230V AC loads, the relay contacts are rated at 250 VAC, 7 A.
The recommended separation distance between tracks to limit the effects of creepage depends on the application and the standards you need to comply with. In a clean and dry environment, the separation distance can be quite small; in a dirty and damp environment it needs to be much bigger. The layout of the relay card is designed for use in clean and dry environments.

Clemens Valens (Elektor Labs)

Dual-Anode MOSFET Thyristor

ElektorLabs 3/2019, p. 15 (190017)

UPDATE. This article is attributed solely to Michael Shustov but we should have acknowledged the work of the co-author Andrey Shustov also.

Dual-Anode MOSFET Thyristor

ElektorLabs 3/2019, p. 15 (190017)

I noticed that this article offers a high-voltage and low-voltage version of the circuit. I don't like to be pedantic but according to the International Electrotechnical Commission (IEC) AC voltage <1000 VAC is classified as 'Low voltage' while the 40 VAC version of the circuit should carry the label 'Extra-low voltage'.

Peter Bitzer

Can't argue with that… many thanks for your observation.

Ralf Schmiedel (Technical Support)



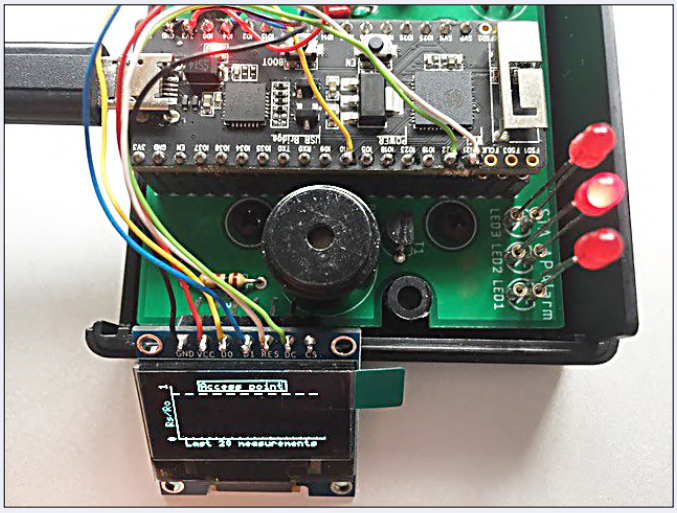
Air Pollution Monitor

Elektor 2/2019, p. 90 (170182)

FEEDBACK. I have an improvement suggestion for this article. Although it is shown in the circuit diagram it isn't explained in detail in the text: With the described configuration and the associated sketch, the LEDs cannot operate in parallel with the OLED display. There really is no need to use the same GPIOs to drive both the three LEDs and the OLED; the ESP32 Pico Kit has enough free GPIOs to control them both independently. I rebuilt the monitor and made a few changes so that I can run the LEDs and the display at the same time.

To make the changes it's necessary to assign the OLED pins to some different connections of the ESP32, the wiring to the LEDs is in the form of PCB tracks so these can't really be changed without making a mess. I chose to use IO4 for the display DAT (data) signal, IO22 for the display DC and IO21 for RST on the display (the display clock signal is unchanged from the original circuit). The sketch must of course be changed accordingly:

```
//pin definitions
#define gasSensorPin 35
#define btn 9
#define alarmPin 19
#define dispClk 10
#define dispDat 4
#define dispD_C 22
#define dispRst 21
#define STALed 5
#define APLed 18
#define alarmLed 23
```



Once done, it's best to go through the entire sketch and search for occurrences of `STALed`, `APLed` and `alarmLed`. In these positions where the commands lines `#ifndef USEDISPLAY ... #endif` and in the `#else`-branch of `#ifdef USEDISPLAY ...` and its terminating `#endif`, remove these. This is necessary in a couple of places in *tgs2600.ino* and *network.ino.*

Once you have taken care of the changes, you should see the display operating in parallel to the LEDs which indicate the correct status of the network connection and the alarm.

The wires to the OLED can be easily soldered to the ESP pins.

Hans Schneider

Many thanks for your suggestions and improvements.

Ralf Schmiedel (Technical Support)

180573-D-02

Equipping Your Maker Lab…

…doesn't have to cost much thanks to us!

Once you have taken your first steps along the electronics path you will have a need for test and measurement equipment. And this doesn't need (in the beginning at least) to be very expensive semi-professional 'stuff'.



OWON XDS3064E four-channel 'scope with Touch Screen

The XDS series oscilloscopes from the Chinese brand OWON has a number of features that you do not (yet) find in comparable models from other brands, and all that for very sharp pricing. Here we take a brief look at the very affordable 4-channel XDS3064E, which has a whole lot to offer for not even € 400.
When unpacking this oscilloscope, its large display immediately catches the eye, with a resolution of 800x600 pixels plus touch screen functionality. This is not a common feature in this price category. After switching on this first good impression is confirmed: a nice screen with a good viewing angle.
As already mentioned, this is a 4-channel 'scope with an input bandwidth of 60 MHz and with a sampling rate of 1 GS/s. While it is true that the sampling rate drops to a half or a quarter when multiple channels are used, this is typical for this price category. The instrument is supplied with four probes, a mains power cable, a USB cable and a brief getting-started manual. A comprehensive manual is available as a PDF.
Becoming familiar with the operation of the XDS3064E goes quite quickly. However, a few of the functions require some further explanation and it is then a good idea to consult the full manual. The sensitivity of the input channels can be set down to 1 mV/div (bandwidth limited). The noise of the input stages remains quite small with this, otherwise such a high sensitivity would make no sense. With the Auto-set button you can automatically search for the appropriate settings for a clear display of the input signal. The Math button does not only allow operations to be performed on the signals, but it is also possible to enter equations yourself. The XDS3064E also has an FFT function.
In conclusion we can state that if an input bandwidth of 60 MHz is sufficient for you then this instrument will give you a great deal of measurement enjoyment.

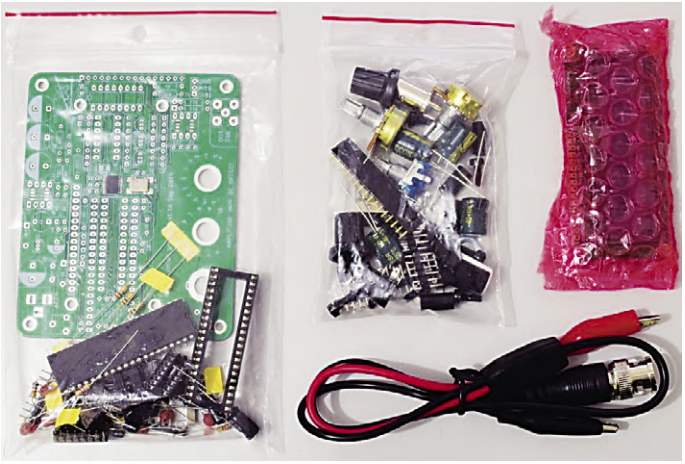
@ WWW.ELEKTOR.COM

→ OWON XDS3064E Four-Channel Oscilloscope
www.elektor.com/18829

Function Generator Kit from Pulsar Labs

An oscilloscope (such as the OWON model described above) is indispensable for seeing what an electronic circuit is doing, but such an instrument becomes really useful when we can apply a suitably defined signal at the input of a circuit. And that is where a function generator comes in. Here we take a look at a function generator that we have to build ourselves first: the open-source function generator from Pulsar Labs.
The assembly of the function generator (although it is not explicitly described in the supplied manual) should be straightforward, even for those who are not yet soldering experts. The most important thing is to take your time, thoroughly check





how the components have to be mounted and in the event of doubt, attempt to fit it first and if necessary bend the legs to the correct size. The only two SMD components are already fitted on the board, so there is no need to worry about that. Once the circuit board has been fully assembled and checked (tip: in the event of doubt, let someone else also cast an eye over it), the power supply voltage can be applied and the generator can be tried out.

The generator can generate the three most important wave forms (sine, square and triangle) and for this price (not quite € 60) has a respectable frequency range of 1 Hz – 10 MHz. Extremely practical is also a sweep mode for quickly determining the behaviour of a circuit over a wide frequency range. The maximum output voltage amounts to about 10 V; the offset range amounts to —5 V to +5 V. The distortion of the output sine at 0.7 V and 1 kHz is about 0.1% — not that shabby at all. All in all this is a very nice generator that certainly would not look silly in the lab of a beginning hobbyist.

Hand-held 3D Printer

Is it for tinkering or for learning? Don't torment your brain any longer: 3D Simo kit is both! When you think about 3D-printing you may well have a reasonably complicated and noisy device in mind and not a pen that you can use to draw in three dimensions and various colours. But some time ago, a 3D-tool with small dimensions made its entry into the world of the tinkerer. It reminds you of the classic glue gun.

The electric tool can be compared to a 3D-printer in the same way as a hand-held drill and a column drill press. And much quieter! Or in other words: it is a PLA extruder that you can hold in your hand like a pencil so that you can 3D-print manually.

It is designed as a hand tool, light and easy to use, and is intended for use by enthusiasts of 3D-printing who would like to take matters into their own hand, literally and figuratively. It is both a tool for touching up and finishing as well as creating things manually. The dimensions are 138 x 25 x 21 mm, and it does not weight more than a few tens of grams. It is powered from 5 V via a USB cable (which has to be capable of supplying 1.5 A).

The model described here is also a kit that you have to put together yourself. The main circuit board is an Arduino Nano. The 3D Simo kit is open-source: all the design data is freely available from GitHub (the circuit board files, the 3D-printed parts and even the software). In other words: everything can be changed and can be personalised.



This kit not only gives you a 3D-pen for manually drawing, but is also an object for learning and experimenting. It is therefore certainly also very interesting for use in education. .

180689-D-02

@ WWW.ELEKTOR.COM

→ Pulsar Labs Function Generator Kit Set
www.elektor.com/18653

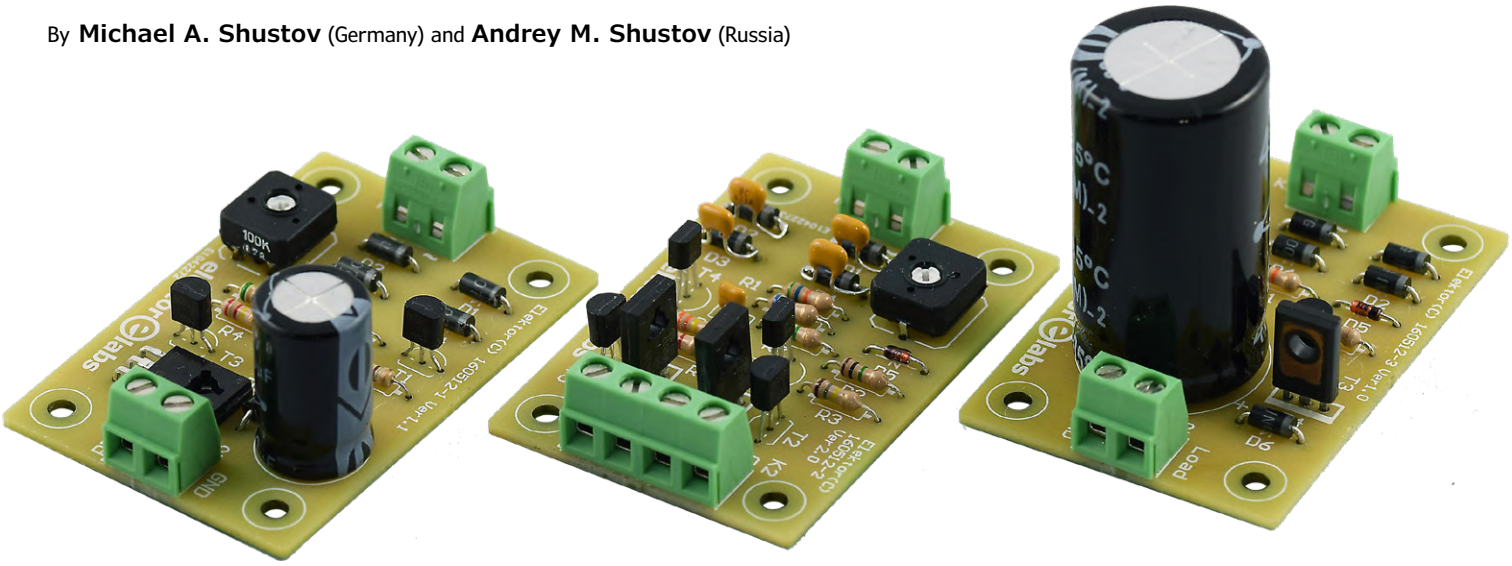
@ WWW.ELEKTOR.COM

→ Simo 3D Pen Kit
www.elektor.com/18867

AC-in, DC-out Baristors

Buildings block for application in power supplies

By **Michael A. Shustov** (Germany) and **Andrey M. Shustov** (Russia)



In industrial and consumer electronics, the use of thyristors and triacs is known to corrupt the AC grid voltage to a degree due to harmonics, noise, bursts and transients being superimposed on what you thought was a clean sinewave from the power outlet. A newly developed composite electronic device called the baristor can help to keep the line voltage clean yet offer good regulating properties. This article describes the baristor's functionality as well as three practical AC-in, DC-out incarnations, on circuit boards

The barrier-resistor element (barrier + resistor = Baristor) is a switching element for power and analogue electronics, whose electrical "input-output" resistance changes abruptly from conducting to non-conducting and vice versa. The desired "state" is obtained when a predetermined threshold value ("barrier") of the input signal is reached. Symbaristors are symmetric baristors intended to operate on alternating current (AC).

Baristor *vs*. SCR

In power electronics, SCRs like thyristors and triacs are known as commutation elements capable of switching from a non-conducting state to a conducting state when their gate electrode is energized. They are switched off when their instantaneous anode voltage is approaching zero. Thyristors and triacs allow cutting off almost any part you want of the AC sinewave signal and so control the effective power fed to the load. But they have some disadvantages:

- low switching performance;
- no turning off without interrupting current through device;
- distortion of AC line sinewave;
- power factor reduction;
- low economic efficiency.

The above drawbacks can be overcome to a large extent by baristors and symbaristors which:

- enable an AC-current signal to be divided into segments of adjustable width and using them according to user requirements;
- you can easily regulate heaters and lamps with thyristor or triac based circuits;
- are suitable for use in power supply units to generate different output voltages.

As a radical thought, consider that the switching in thyristors and triacs occurs on the "horizontal" axis (i.e. with time) while with baristors, switching occurs on the "vertical" axis (i.e. with voltage). This feature of the baristor opens fundamentally new opportunities for its use in communications equipment and power electronics. Baristors are intended to separate signals whose amplitude is above or below a predetermined threshold value (barrier) set by the user.

Types of baristor, and applications

Baristors can be categorized as:

- Controlled and uncontrolled (i.e. with a controlled or an uncontrolled threshold value);
- AC or DC (asymmetric and symmetric baristors — symbaristors);
- Single-channel or multi-channel (multi-level, multi-threshold baristors);
- Switches of type: On/Off/switchover.

Here are the names with the family picture.

High-level baristor: the voltage at the output occurs only in the case when the input voltage level exceeds a certain threshold value.

Low-level baristor: the voltage at its output is present as long as the input voltage level exceeds a certain threshold value. After that, the baristor changes its state and turns-off the load.

Switching single-threshold baristor: as it rises gradually, the input voltage is initially present at the low-level signal output. After exceeding a certain threshold value, the input signal is automatically switched over to the high-level output.

Switching multi-threshold baristor: By continuously increasing/decreasing the input voltage a sequential switching-over of the baristor outputs occurs and thus the input signal is passed on to the baristor outputs. Thereby the input signal being passed-through indicates the corresponding voltage range.

The use of baristors and symbaristors allows distribution or redistribution of AC-power between consumers of electric energy, to suppress higher harmonics of the line voltage, to increase (i.e. correct) the power factor, to use a part of the sinewave rationally that SCRs "throw away". All these features should help to increase the efficiency of electric energy use significantly, as well as to improve economic effectiveness. Barrier-resistor elements can be applied in the following areas:

- miniature economic power supply units with low losses;
- voltage converters and voltage stabilizers;
- protection of electronic components and circuits as well as electric power equipment and communication lines.
- systems for multichannel remote control and communication systems including a HF signal superimposed on a two-wire line;
- measuring instruments and transducers;
- devices for pulse technique;
- isolation, separation or generation of signals;
- analogue electronics for signal separation into components using a user-set threshold value.

If you are interested in knowing more about the origins of the baristor and its theory of operation, get a copy of the authors' book "Electronic Circuits for All" published by Elektor; details are given in the **@ www.elektor.com box**.

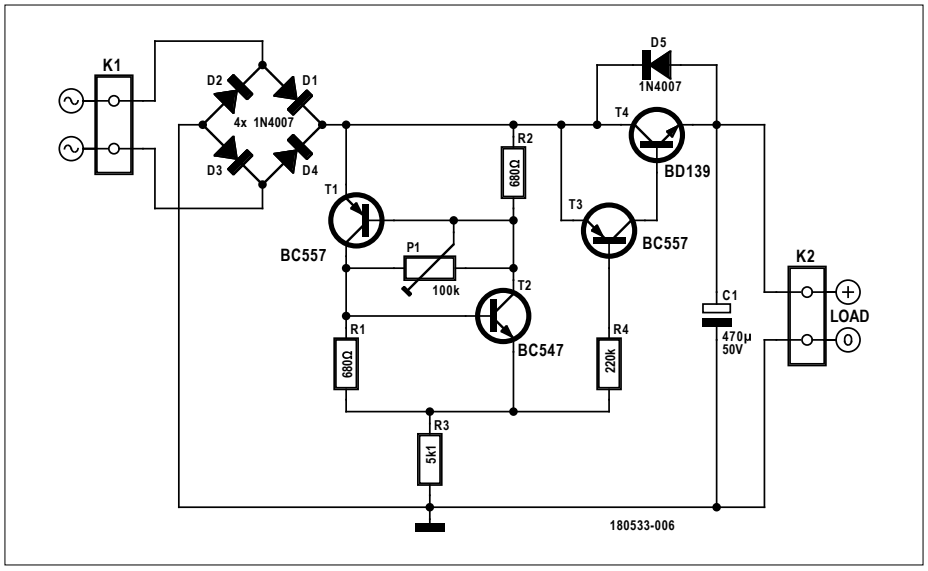


Figure 1. Basic ('starter') version of a single-threshold baristor as developed out of a concept discussed in the book "Electronic Cicuits for All".

Performance, Single-threshold Baristor

Reference: circuit in Figure 1.
Some measurements results using 150-Ω load and fixed setting of P1:

AC in	DC out
12 V	11.4 V
16 V	9.9 V
20 V	9 V
24 V	8.6 V

At 20 VAC input voltage the output voltage was seen to drop from 9 to 6.7 V when the load resistance was halved to 75 Ω. At certain input voltages and settings of P1 the output voltage was observed to fluctuate over time. No-load current can be under 1 mA but also a few mA, depending on the setting of P1, input voltage and load removal.

Baristor #1, the basic one

Moving on to practical incarnations of the baristor, **Figure 1** shows the schematic of a single-threshold baristor post-engineered by Ton Giesberts of Elektor Labs, from a version found in the above-mentioned book.

As long as the voltage at the output of the bridge rectifier D1–D4 (about 16 VAC is used in this case) remains under the threshold set by T1/T2, "composite transistor" T3/T4 is conducting. The actual voltage level from D1–D4 minus the voltage drop across T4 gets stored in buffer capacitor C1. If the input voltage exceeds the set threshold, T4 blocks and the output voltage does not increase further. This way, depending on the threshold level set by T1/T2, the DC output voltage can be set on P1. Replacing P1 with an optocoupler or a JFET permits the output voltage to be controlled by another circuit. If the applied AC voltage is higher than 16 V or so, resistors R3 and R4 should be changed accordingly. Depending on the values of R3 and R4 and other conditions, the current-voltage characteristic of the cir-

COMPONENT LIST

Single-threshold Baristor / PCB 160512-1 v.1.1 / Circuit Figure 1.

Resistors

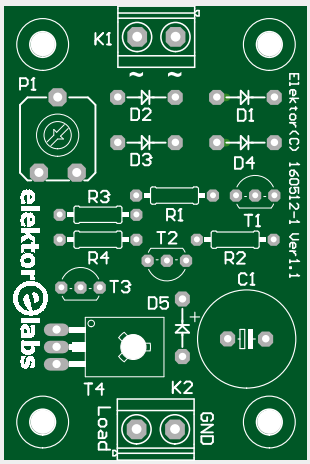
R1,R2 = 680Ω 5%, 0.25W
R3 = 5.1kΩ 1%, 0.25W
R4 = 220kΩ 5%, 0.25 W
P1 = trimpot, 100kΩ 20%, 0.15W, CB10LV104M Citec-TE Connectivity

Capacitor

C1 = 470µF 20%, 50V, diam. 13mm, pitch 5mm

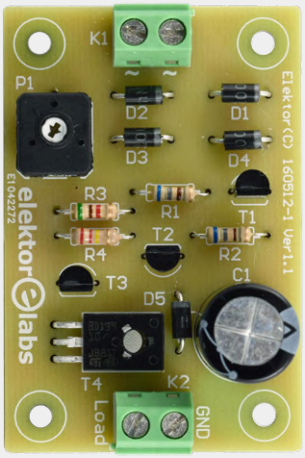
Semiconductors

D1,D2,D3,D4,D5 = 1N4007
T1,T3 = BC557B
T2 = BC547B
T4 = BD139



Miscellaneous

K1,K2 = 2-way PCB screw terminal block, 0.2" (5.08mm) e.g. Phoenix Contact MKDSN 1,5/ 2-5,08



PCB 160512-1 v1.1 from Elektor Store

cuit around T1/T2 varies from "thyristor" to "Zener diode". The problem with this is… it's not really a Zener diode. With T1 and T2 not conducting, current still flows through R2-P1-R1-R3. So if P1 is too low in value or R3 too high, T3 may switch on or off, i.e. there's no real threshold and the output voltage will vary with the AC input voltage. Also, some very low frequency instability can occur at certain settings of P1 in combination with different AC input voltages. Worse, the output impedance of the transformer affects the operation and stability of the circuit. A 10-µF capacitor from the base of T4 to ground can improve circuit stability. A change in load resistance will change the output voltage.

Some salient measurement results obtained with a practical construction of the circuit are given in the text box**: Performance, Single-threshold Baristor.**

Enhanced Single-threshold Baristor

The circuit in **Figure 2** is again based on an outline found in the 'Electronic Circuits for All' book.

Its purpose and intent isn't so much a fully working device as a circuit that can be modified to fit your own applications or just give new ideas or be part of a solution for other purposes. Sure, the PCB designed by Elktor Labs for this circuit does make testing and experimenting easier but the circuit is simple enough to be tested on a breadboard.

A positive feedback network R7–C5 was added to suppress erroneous switching and to enhance switching of the output stages. This makes the base of T1 act like the input of a Schmitt-trigger. Due to the hysteresis, which depends on the AC voltage applied to K1, the phase set on P1 changes with the level of the input voltage. Depending on the application and AC voltage used the values of some components are subject to tweaking, for which simulation software like LTSpice comes in handy. Using a voltage source instead of a "real" transformer in the simulation will give false results. Every transformer has a complex output impedance consisting mainly of a combination of a resistance and an inductance. As a result, the moment the Baristor switches, the input voltage changes and can cause instability, especially when the gain of T1 is high.

The original circuit found in the book used high voltage Darlington transistors type BC373 which sadly proved hard to find. The specifications are: 80 V / 1 A in a TO92 case style. The parameter hFE at 100 mA is quoted as >20000. Fortunately there are quite a few replacement types that are still available, including BCX38C (60 V / 0.8 A); KSP13B (30 V / 0.5 A); MPSA14, (30 V / 0.5 A); MPSA29 (100 V / 0.5 A); ZTX603 (80 V / 1 A); ZTX605 (120 V / 1 A); ZTX614 (100 V / 0.8 A); BC517-D74Z (30 V / 1.2 A but with its collector and emitter connections swapped!

But be careful using a TO92 packaged device for T3 and T6 — due to the limited power dissipation of this housing it's wiser to use a larger case style. In the end we used the good old BD139 in combination with a standard BC547C to make a discrete Darlington replacement for the originally used BC373. The BD139 has a SOT32 (TO126) case style and allows a maximum power dissipation of 1.25 W without additional cooling (i.e. the junction will be 150 °C at room temperature, 25 °C). Never operate a transistor at a junction temperature of 150 °C, keep it well below 70 °C if you don't want to reduce its lifespan. And why not use a BD679 Darlington to reduce the number of parts, you might ask? Well, almost all Power Darlington's have internal resistors in parallel with the base-emitter of both transistors. These inevitably cause the gain of the transistor to vary with the collector current, while at low currents the gain is much lower than that of Darlingtons without internal resistors. Back to the circuit diagram, the maximum voltage drop from K1 to K2 or K3 is about 4 V at 100 mA. This is due to D1–D4 and a small voltage drop across R4 and R6 and the base-emitter voltages of T2/T3 and T5/T6. Hence without extra cooling of T3/T6 the maximum output current should be limited to a few 100 mA.

Removing D5 will increase the range of P1 and has more effect at low input voltages. To change the range of P1 at high

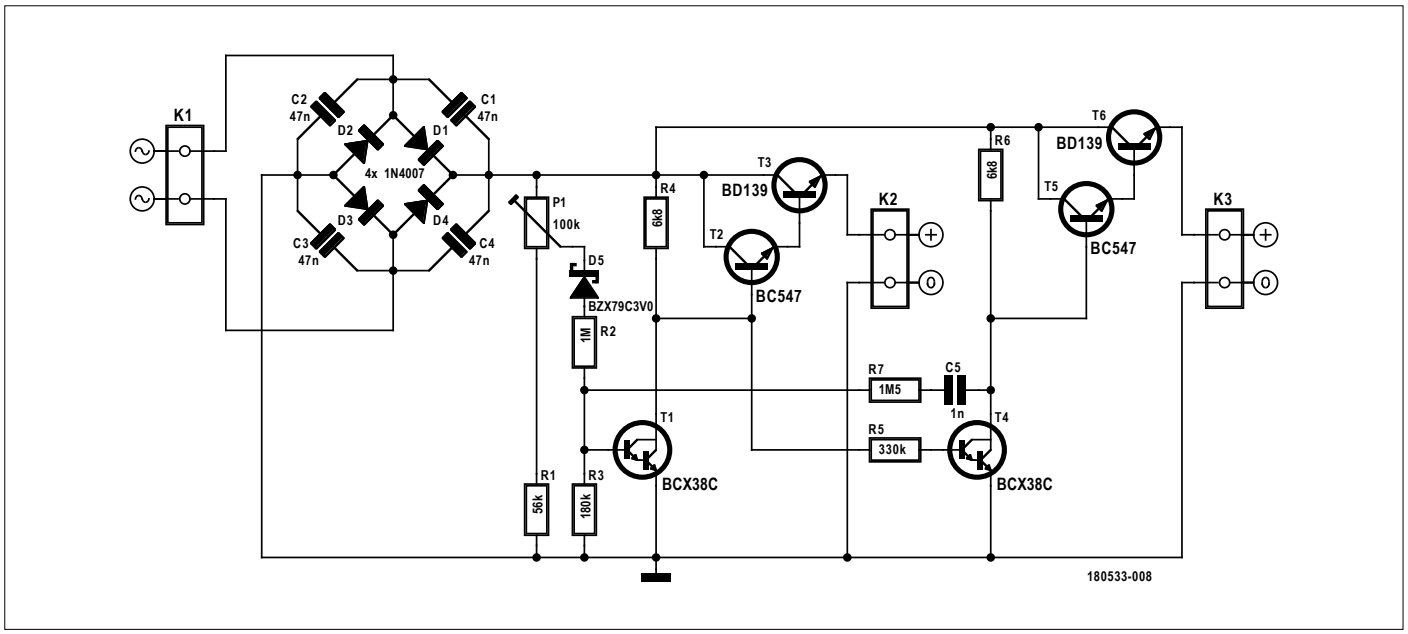


Figure 2. This version of the baristor has enhancements and refinements compared to the basic version. It should offer many possibilities for experimenting and adapting to your requirements.

Performance, Single-threshold Baristor, Enhanced Version

Reference: circuit in Figure 2.
An example of maximum phase angles at minimum and maximum setting of P1 (18 V at K1):

- P1 fully clockwise: 29° - 148°
- P2 fully anticlockwise: 79° - 126°
- No load current: 1.71 mA (12 V at K1)
- No load current: 3.75 mA (24 V at K1)

◦

With equal load impedances on K2 and K3:

- Input current: (U_{in} – 3.2 V) / R_{load}, measured at 300 mA (K1) and 75-Ω loads.
- Input current: (U_{in} – 3 V) / R_{load}, measured at 120 mA (K1) and 75-Ω loads.

With two identical loads connected to K2 and K3 the input current is virtually independent of the setting of P1 and varies only slightly, a few percent, due to component tolerances.

COMPONENT LIST

Single-threshold Baristor, Enhanced Version / PCB 160512-2 v.2.1 / Circuit Figure 2.

Resistors

R1 = 56kΩ 5%, 0.25W
R2 = 1MΩ 5%, 0.25W
R3 = 180kΩ 5%, 0.25W
R4,R6 = 6.8kΩ 5%, 0.25W
R5 = 330kΩ 5%, 0.25W
R7 = 1.5MΩ 5%, 0.25W, 250V
P1 = trimpot, 100kΩ

Capacitors

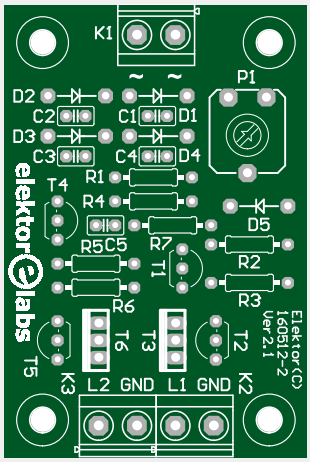
C1,C2,C3,C4 = 47nF, 50V, ceramic X7R, 0.1" pitch
C5 = 1nF, 100V, ceramic X7R, 0.1" pitch

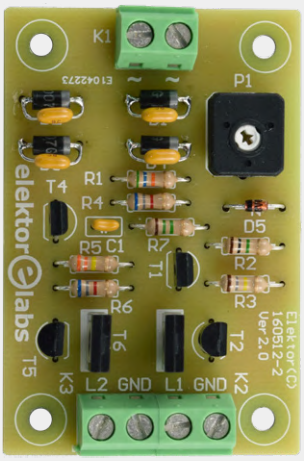
Semiconductors

D1,D2,D3,D4 = 1N4007
D5 = BZX79-C3V0, 3V, 500mW
T1,T4 = BCX38C, Darlington, 60V, 0.8A
T2,T5 = BC547C
T3,T6 = BD139

Miscellaneous

K1,K2,K3 = 2-way PCB screw terminal block, 0.2" pitch (5.08mm), MKDSN 1,5/ 2-5,08
PCB no. 160512-2 v2.1 from Elektor Store





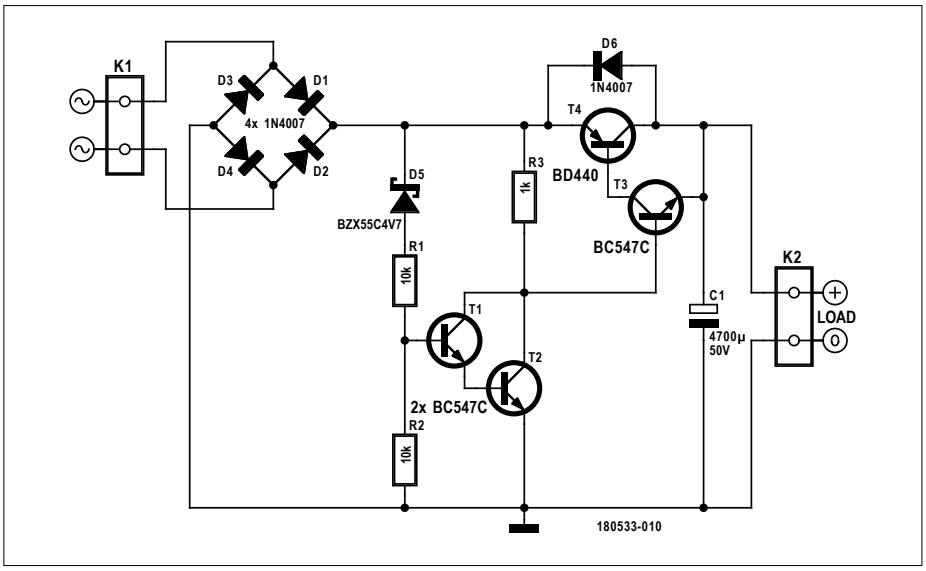


Figure 3. This baristor version is the result of seeking a tradeoff between component count (low) and stabilisation (high).

input voltages consider changing R1 and R3 also, and maybe the working voltage of D5. But due to the low current flowing through D5 the voltage drop across it is less than the rated working voltage. We have again grilled this circuit in the Labs and the results with comments are in the **Performance, Single-threshold Baristor, Enhanced Version** text box**.** Our test setup used a variable autotransformer, also known under the trademark name Variac. It's used to vary the primary voltage of a "safety transformer" type FL30/12 from Block. This has two primary windings of 115 V and two secondary windings of 12 V able to supply 1.25 A each. This way a safe test setup is guaranteed. But be very careful in this setup. The maximum output voltage of the autotransformer is typically higher than the input voltage. Labs used one that is still specified for use with the old 220 V mains voltage. The output is specified as 250 V / 2 A max. At the present time, with the mains voltage upped to 230 V nominally, the no-load output voltage will be even higher, like 261 V. But it's even worse when measuring the actual output voltage, a whopping 274 V was found with the mains voltage reaching 239 V at the test location. In any case, never adjust the autotransformer to its maximum voltage in this test setup. This will most certainly destroy the safety transformer connected to it.

In a first test a small transformer was used and R7/C5 were enough to get rid of the erroneous switching. But using the test setup as described above this wasn't enough. The four diodes of the rectifier had to be decoupled with 4 off 47 nF anti-rattle capacitors.

COMPONENT LIST

Dressed-Down Baristor / PCB 160512-3 v.1.0 / Circuit Figure 3.

Resistors

R1,R2 = 10kΩ 5%, 0.25W
R3 = 1kΩ 5%, 0.25W

Capacitor

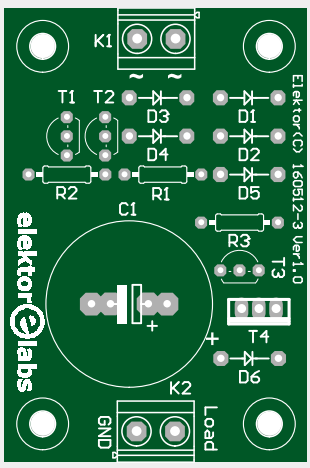
C1 = 4700µF 20%, 50V, diam. 22mm max., pitch 5/7.5/10mm

Semiconductors

D1,D2,D3,D4,D6 = 1N4007
D5 = BZX55C4V7, 4.7V, 500mW
T1,T2,T3 = BC547C
T4 = BD440 (PNP, 60V, 4A, 7A pulse, TO-126)

Miscellaneous

K1,K2 = 2-way PCB terminal block, 0.2" pitch (5.08mm) 630V
PCB no. 160512-3 v1.0 from Elektor Store





Parts count down, stabilisation up

To give an idea of further enhancements and optimizations within easy reach, a petite baristor circuit was designed that produces a DC output voltage less dependent on load and AC input voltage while using a minimum of parts. The principle is simple. The output stage acts as a switch and needs to be turned off as soon as the input voltage exceeds a certain threshold. The latter is set mainly by Zener diode D5. Here R1 limits the current at higher input voltages while R2 makes sure some current flows through the Zener before discrete Darlington T1/T2 turns on. T1/T2 pulls the base of composite transistor T3/T4 to ground, thereby turning off the output switch. To minimize the drop across T4 a combination of an NPN small signal and a PNP power transistor is used, saving a few tenths of volts. To make the drop due to the base current through R3 even lower, the value of R3 is a mere 1 kΩ. This latter means a .25-watt resistor

will dissipate its maximum permissible power at 21 V input voltage. Any measurements with input voltages above 21 V will exceed the safe specs for R3. In that case use a .5- or .6-watt resistor in lieu of .25 watt. There are also types with a 1-watt specification in this package. Mount such resistors slightly above the PCB. Dissipation will be largest with no load.

Since the T3/T4 will be turned off permanently the voltage across R3 is almost all the input voltage, minus the voltage lost across the bridge rectifier. The fact that T1/T2 turns on above a certain voltage doesn't change much. With a Zener of 4.7 V (.5 W) the output voltage is approximately 4.4 V with a 150-Ω load. A 10-Ω load reduces the output voltage to just under 4 V. No-load voltage is about 4.67 V. Originally we used a good old BD140 and at low output current it worked well. But with a higher load, such as 10-Ω resistor, it almost immediately short circuited. Our test setup is able to deliver a nominal 1.25 A AC RMS current, so peak even more. A BD140 can handle 1.5 A peak currents which especially with a large capacitor at the output is not enough. It got replaced with a BD440 from old stock but this type is still available (BD440S, Mouser # 512-BD440S). The BD440 is a 60 V/4 A (7-A pulse) PNP power transistor. There are other suitable candidates in a TO-126 (SOT-32) package to replace T4, like 2N5195 (PNP, 80 V, 4 A), BD438 (45 V, 4 A) or BD442G (80 V, 4 A). Depending on the manufacturer lead spacing can differ somewhat, instead of 2.29 mm, 2.39 occurs (TO−225AA). Also the transistor package can be a little bigger like 14.2 x 8 mm² instead of the original TO-126 dimensions of 11.1 x 7.8 mm. Fortunately all of these should still fit on the PCB. The footprint for (large) buffer capacitor C1 is designed for capacitors with a lead spacing of 5, 7.5 and 10 mm and a maximum can diameter of 22 mm (no snap-in pins, just straight leads).

The output voltage can be made adjustable by replacing R2 by a potentiometer. The output is indeed adjustable then but at higher voltages it gets increasingly dependent on the input voltage. The voltage across R1 will increase more with a lower value of P1 and becomes more dominant than the Zener voltage. To set a different output voltage it's best to change the Zener voltage. If surges at the bridge rectifier occur these can be reduced by inserting a diode between D5 and R3.

And again we tested the circuit in the Elektor Labs, for some results see the text box: **Performance, Dressed Down Baristor.**

Performance, Dressed Down Baristor

Reference: circuit in Figure 3.
Measurements results with T4 = BD440 and 150-Ω load:

U_{in} [V]	I_{in} [mA]	U_{out} [V]	U_{ripple} [mV]
AC	AC	DC	AC_{pp}
5	44	4.28	24
6	64	4.48	56
7	74	4.46	60
8	80	4.44	60
9	85	4.43	60
10	89	4.42	60
11	92	4.42	60
12	94	4.41	60
13	96	4.41	60
14	98	4.40	60
15	101	4.39	60
16	102	4.38	60
17	104	4.37	60
18	106	4.36	60
19	109	4.36	60
20	110	4.35	60
21	110	4.34	60
22	111	4.33	60
23	112	4.33	60
24	113	4.32	60

At low loads most input current is due to R3 being pulled to ground by T1/T2. For use with very low loads consider increasing the value of R3. The drop across T4 may increase but it may increase overall efficiency. Something to experiment with.

Another quick test with a much higher load of 10 Ω was also performed. At 12 V input voltage the input current for a 10-Ω load equals 0.57 A. The output voltage was 3.99 V. But T4 ran hot without cooling! Efficiency is not great but is more complex than multiplying voltages and currents since currents and voltage inside a baristor are far from sinusoidal.

To your irons!

Although all three circuits were found to be working, they are experimental and not to be considered ready-engineered projects for use in industrial applications. Remember, they are not current limited or protected against overloading. Also, the case temperature of the output power transistors should be monitored at higher loads and input voltages. Play it safe and attach a small aluminum plate or similar contraption to the power transistor(s) casing for cooling.

(180533-01)



@ WWW.ELEKTOR.COM

→ Electronic Circuits for All, by M.A. Shustov and A.M Shustov.
www.elektor.com/electronic-circuits-for-all

→ Single-threshold Baristor, bare PCB no. 160512-1 v.1
www.elektor.com/160512-1

→ Single-threshold Baristor, Enhanced Version, bare PCB 160512-2 v.2.1
www.elektor.com/160512-2

→ Dressed-Down Baristor, bare PCB no. 160512-3 v.1.0
www.elektor.com/160512-2

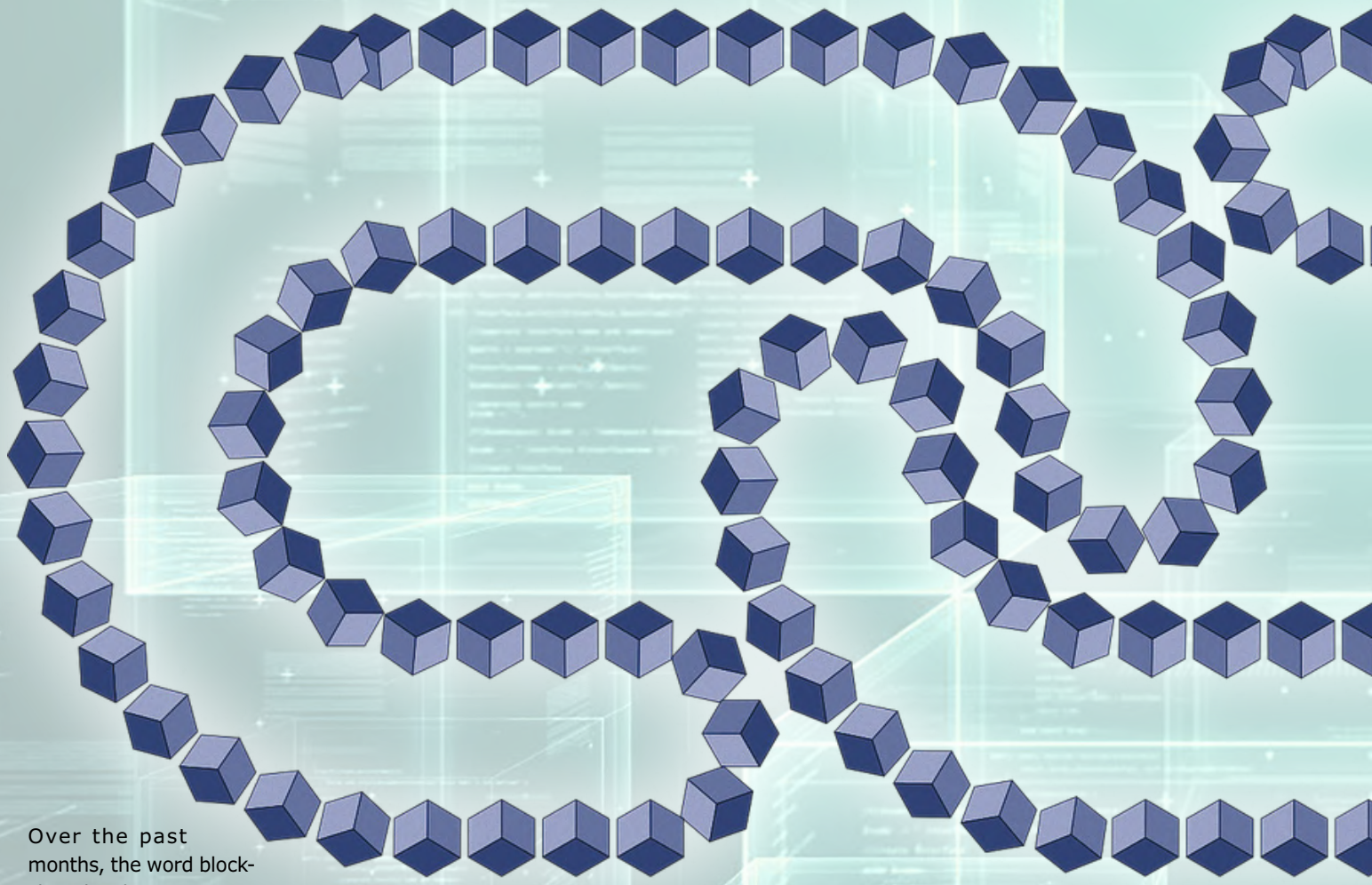
HomeLab Helicopter

Wondrous Things Electronic Spotted From Above

By **Clemens Valens** (Elektor Labs)

Blockchains,

the Universe, and Travelling Through Time



Over the past months, the word blockchain has become an important buzzword that is said to help selling content. Mention blockchain and you have an audience. The fact that you are reading this proves my point.

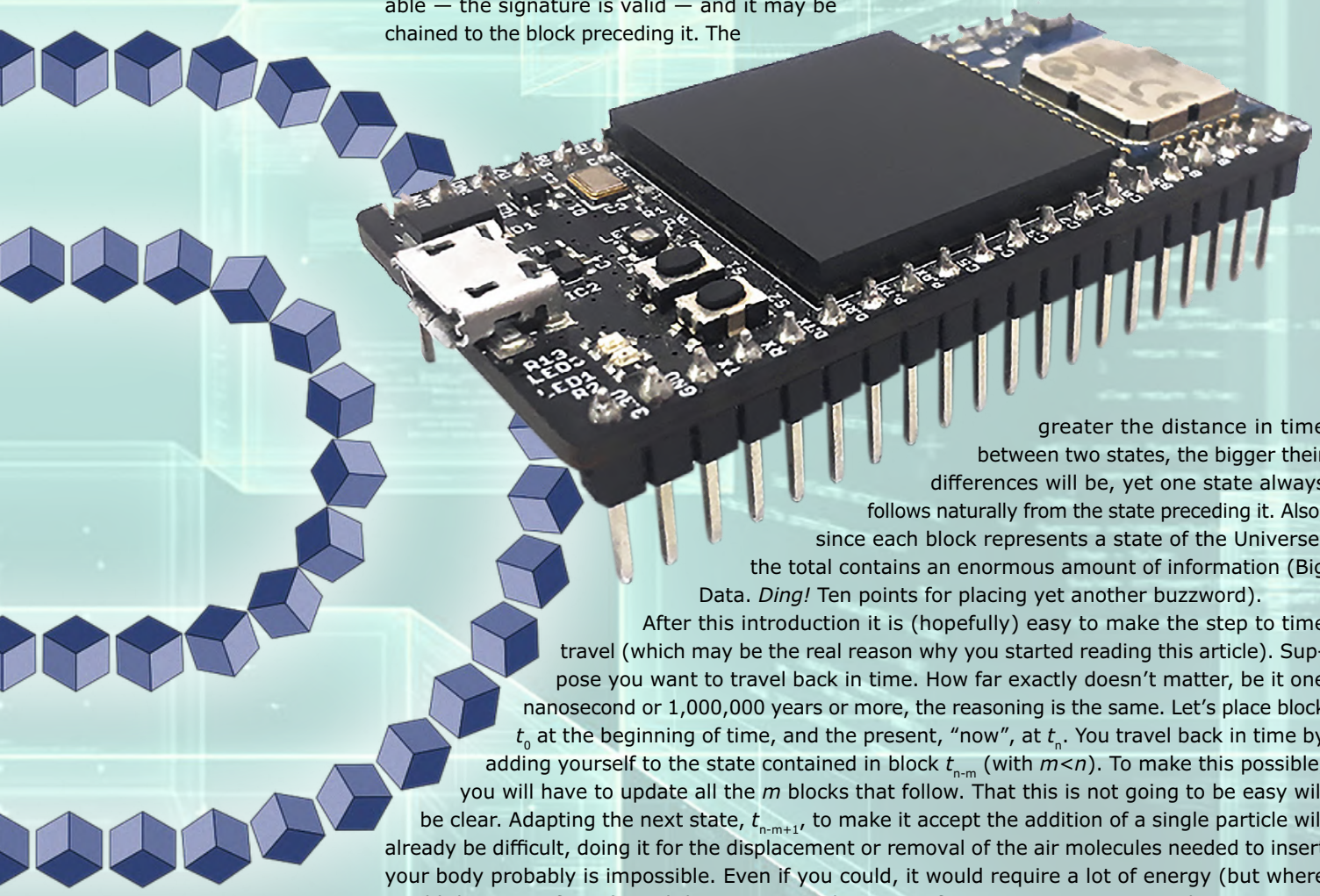
But what is a blockchain? Loosely after Wikipedia we can define it as a chain of blocks where each block contains a signature (a 'hash') of the (contents of the) block preceding it, a timestamp, and transaction data. Calculating the signature of a block is complex, making this an expensive and time-consuming operation. Furthermore, every new signature must be accepted by several verifiers. Modifying the contents of a block somewhere in the chain requires updating the signatures of all the blocks that follow, to prevent the blockchain from becoming invalid. If updating one signature is "complicated" — imagine the effort needed to update several hundred or thousand of them! A blockchain is therefore quite resistant to modification of the data it contains.

Soon on a breadboard near you

Even though blockchains tend to be associated with cryptocurrencies, the Internet of Things (IoT, buzzword 2) is another interesting domain for blockchain-based secure data communication. By the time you read this, Egypt-based startup Elkrem should have terminated — successfully? — their Kickstarter campaign for their Blockchain IoT Development Board.

In the real world

Some time ago it dawned on me that it is possible to transpose the concept of virtual blockchains to the real world. Imagine that the state of the Universe at some moment, say t_0, can be captured in a block. The state of the Universe one moment later, at t_1, is almost identical to the previous state but not completely as some particles have moved. Because the particle movements were guided by the Laws of the Universe, its new state is acceptable — the signature is valid — and it may be chained to the block preceding it. The



greater the distance in time between two states, the bigger their differences will be, yet one state always follows naturally from the state preceding it. Also, since each block represents a state of the Universe, the total contains an enormous amount of information (Big Data. *Ding!* Ten points for placing yet another buzzword).

After this introduction it is (hopefully) easy to make the step to time travel (which may be the real reason why you started reading this article). Suppose you want to travel back in time. How far exactly doesn't matter, be it one nanosecond or 1,000,000 years or more, the reasoning is the same. Let's place block t_0 at the beginning of time, and the present, "now", at t_n. You travel back in time by adding yourself to the state contained in block t_{n-m} (with $m<n$). To make this possible, you will have to update all the m blocks that follow. That this is not going to be easy will be clear. Adapting the next state, t_{n-m+1}, to make it accept the addition of a single particle will already be difficult, doing it for the displacement or removal of the air molecules needed to insert your body probably is impossible. Even if you could, it would require a lot of energy (but where would that come from?). And then you must do it again for states t_{n-m+2}, t_{n-m+3}, and so on.

You could limit the amount of calculations needed by keeping your presence in the past very short, a few states only, so that the disturbances you created can die out quickly, allowing later states to remain unmodified. Your impact on the past must be as little as possible.

Even if we suppose that you can solve these problems, then you will have to face similar issues when you return to the present. Any dust particles or memories — a state change of the brain — you picked up during your visit in the past will have to be accounted for in the states that follow your visit. To avoid this mountain of work, you should not bring back anything from the past to the present. Unfortunately, this renders time travel rather pointless.

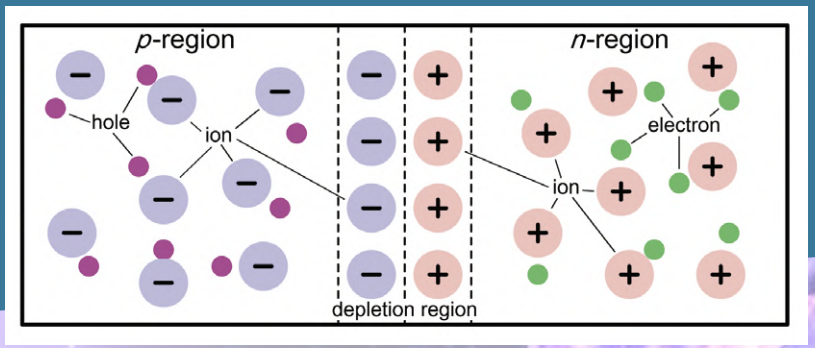
Wormholes



I hear you clamour, what about wormholes? According to some theories, wormholes might allow for travelling through time. Well, blockchains can have wormholes too as nothing forbids chaining more than one block to the same block. In blockchain terminology this is called a fork and the result is two or more paths forward. (Homework: answer the question, "Can the Universe be forked?" in 500 words.) One of these paths could loop back to a block earlier in the chain. The latter does not have to link to the alternative previous block. However, even if a wormhole would let you go back to a block in the past, entering that block would create the same problems as the ones mentioned above.

So, what about travelling to the future? Well, that would mean entering a state t_{n+f} that not yet exists. (If it does exist, we would be living in the past. Following the blockchain analogy this would imply that all our actions are already known, otherwise the future would be invalid. So long, free will.) You will have to create this state together with the path that attaches it to the present state t_n. How you configure the states that make up this path is up to you, as long as they form a valid blockchain. This implies that the future only depends on your imagination. Simply closing your eyes and dreaming away would be a much easier and less energy-consuming yet equally valid alternative for travelling to the future.

The crystal $NaSn_2As_2$ can be both electron-holder and hole-holder, potentially eliminating the need for multiple layers in semiconductors.





Must-have Homelab Tool

Every once in a while you want to measure a voltage or current in a place where it is difficult to look at the multimeter's display while holding the probes in place. A Bluetooth multimeter like the one shown here can help out in such situations. A connected device like a smartphone acts as a mobile display that you put in a place where you can actually look at it. If even this is not possible, then the smartphone can read out loud the captured values while you concentrate on the probes. Even though these handy functions alone make such a multimeter qualify as a must-have homelab tool, there is more. Data logging and graphing and even multiple meters displayed on one phone. Cool, huh?

www.elektor.com/owon-ow16b-digital-multimeter-with-bluetooth

Smoking Really Is Dangerous

Okay, so you knew that already. Maybe you replaced your traditional tobacco-based cigarettes by a hopefully safer e-cigarette? The long-term health risks of the e-liquids used by these devices are still unknown, but you probably knew that too. What you may not know yet is that in the French village of Plougoulm a man got burned when the battery of his e-cigarette went up in smoke after short-circuiting with his car keys in his trouser pocket. According to the victim, after hearing an explosive sound, a 40-centimetre long flame shot out of his pants. When trying to remove the burning (!) e-cigarette, the man injured his hand. (I guess he was lucky to get away with just that.) Similar incidents involving coins have been reported too.

But there is worse. In Fort Worth, Texas, USA a man got killed when his e-cigarette exploded in his face and cut a major artery in his neck.

eLABS PROJECT

ESP32 as a Time Server

ESP32 plus RTC plus GPS plus display equals an NTP server on your own LAN

By **Mathias Claußen** (Elektor Labs)

Sometimes an NTP server on your own local network can come in very handy. As well as situations where your network is cut off from the Internet but various pieces of software need a time reference, there are also cases where you are developing a project that requires the ability to fetch time data over the Internet, but you would ideally like to create custom NTP data. As the following guide shows, achieving all this and more is (relatively) easy.

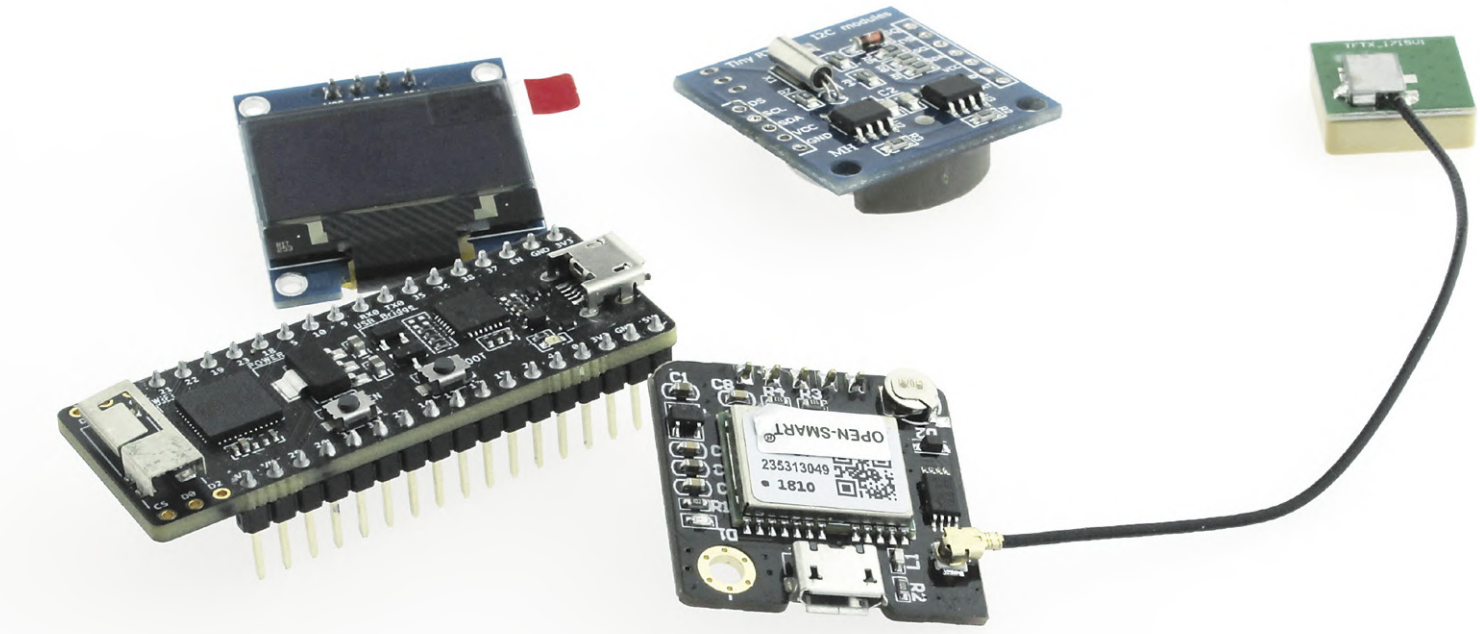


Figure 1. The NTP server consists of these four modules.

The executive summary is that an NTP server is a device that sits at a given Internet address and, when sent a suitable request, returns the exact time using the Network Time Protocol (NTP). If you would like to know the details of how the protocol works, Wikipedia and the like are your friends.

In order for PCs and other small computers that have an Internet connection can know the exact time (and date) immediately that they are switched on, they almost universally include a battery-backed real-time clock (RTC). This supplies the time and date information to the OS, which in turn provides it to software running on the device. The information is needed for much more than just the calendar app: many of the essential operations of the machine, such as ensuring that files that are created or modified are correctly time-stamped, rely on it.

However, the time needs to remain accurate over the long term, and not depend on the accuracy of the RTC's internal reference (a quartz crystal). It is therefore a good idea for the computer's operating system to calibrate its RTC time periodically against an external reference. And that is where NTP comes in.

It is of course possible to calibrate the time using the DCF77 or GPS signals, but using a bit of software to interrogate an NTP server is simpler and cheaper than specially-designed hardware. And furthermore both these time sources have their own disadvantages: the DCF77 transmitter near Frankfurt in Germany

has a considerable, but not infinite, range and is not easy to receive overseas. While GPS can be received around the globe, reception is often not satisfactory inside buildings as many are built from materials which severely attenuate the high-frequency satellite signals. And so NTP is the standard for time calibration.

A DIY time server?

The question remains: why bother making your own time server when there are plentiful public, freely-accessible NTP servers on the Internet?

As we mentioned in the introduction, there are situations where the not inconsiderable effort of building your own NTP server makes a lot of sense. There are networks that, for very good reasons, are not connected to the Internet; and such isolated networks will still as a rule include computers and other devices that need to know the time. Setting the clocks on all the network nodes manually whenever needed, and having to adjust them every month or two, is no fun at all.

Also, anyone developing devices and/or software tools that need to fetch time data over NTP will find it extremely convenient to be able to control the data sent out by the NTP server for testing purposes. Indeed, in past issues of this magazine we have described several projects that use NTP data. Moreover, access to a device located on a local network is quicker and more reliable than access over the Internet. Another application for a local NTP server is when operating a large number of diverse connected devices, which must operate together as synchronously as possible.

So there are many reasons why one might want to build and operate one's own NTP server, and more will probably occur to you. The hardware required is nowadays available at very low cost, and the server can easily be built from ready-made modules. For our purposes the processing power available in an ESP32 module is more than enough, and this module also provides for a wireless connection to the network which allows it to be physically placed almost anywhere convenient. The actual time information for our NTP server project will come from the GPS signal, for which low-cost ready-made modules are again available. We will add a real-time clock for backup in case the GPS signal cannot be received or if there is interference, and, so that we can monitor the state of the server continuously, we also provide a small display. Last, and by no means least, we need the software to bring the server to life, and we will look at that below.

History

Our mini NTP server was first constructed in 2018 for an exhibition in France, where Elektor had a stand. This first incarnation was based on an ESP8266; the other ingredients were just a DS3231 RTC chip and a small 0.96 inch OLED display. The server allowed all the items on display on the stand to receive time data even when no Internet connection was available or if the connection was too poor. This prototype was also pressed into service at Electronica 2018 in Munich, Germany. Subsequently in the Elektor labs it was used as a source of time information for various clock projects and Raspberry Pi-based designs. For this the ability to manipulate the NTP data came in very handy.

The first version of the server was fitted with an RTC chip that showed considerably poorer timekeeping accuracy than expected from the DS3231's specifications. This rapidly became annoying, as the time had to be reset every month. While that is no problem over the duration of an exhibition, in our labs it rapidly became impractical. And so the NTP server hardware needed an update.

The first change was to replace the ESP8266 module with the more modern ESP32 version, and, to enable automatic time synchronization with the GPS signal, a receiver from the Elektor store was added. The ESP32 module we chose comes with an OLED display already fitted. This display is configured to respond to I²C address 0x7A rather than the more usual 0x78, which allows the possibility of connecting a second external OLED display at address 0x78. The U8G2 library that we used allows different display addresses to be selected, and so we arranged for the built-in display to show date and time (in GMT), the IP address of the NTP server on the local network, and the status of the ESP32 (access point or client). On the second display we showed local time and the GPS position. And finally we found a spare DS3231 module in the lab that we could use. With that, the hardware update was complete.

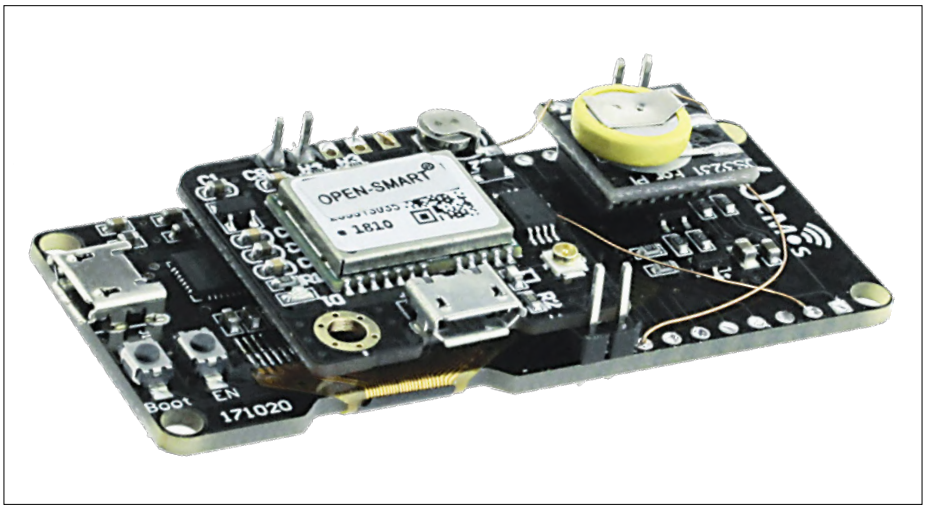


Figure 2. The sandwich comprising the ESP32 module with integrated OLED display, the RTC and GPS modules, is connected up with flying leads.

Modules

To make our new NTP server the above ingredients must be stirred together as shown in **Figure 1**. There are four component modules in total: at the heart of the system is the ESP32 module, and accompanying it are an OLED display with SSD1306 controller, an RTC module and a GPS module. Using the ESP32 module with on-board OLED display saves a bit of space in the design. Together with the DS3231 RTC module and the GPS module it can be assembled into a compact sandwich, as shown in **Figure 2**, for which a suitable enclosure was 3D-printed (see **Figure 3**). The result is a small and, most importantly, mobile NTP server that can be used anywhere that its built-in WLAN can find a connection and where GPS signals can be received.

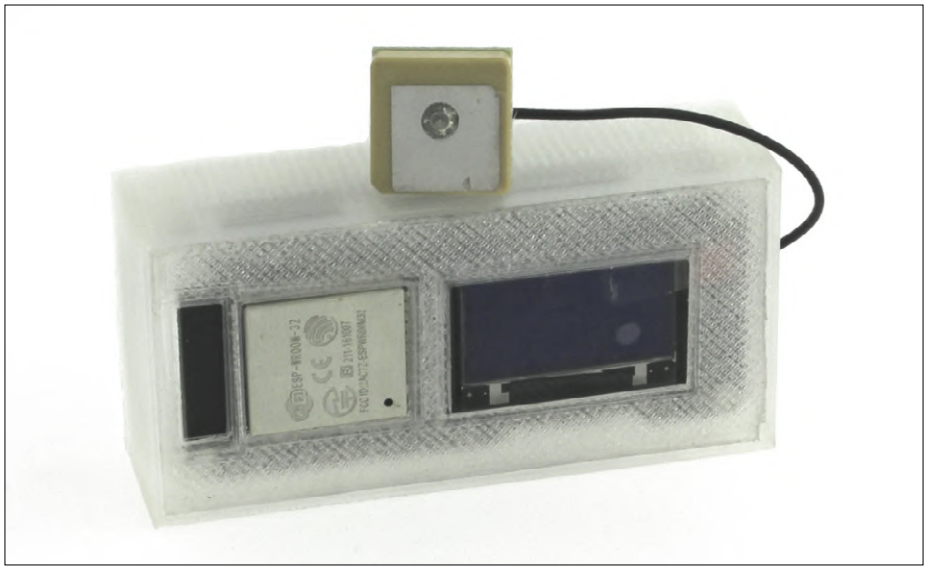


Figure 3. The items of Figure 2 fitted into an enclosure made using a 3D printer.

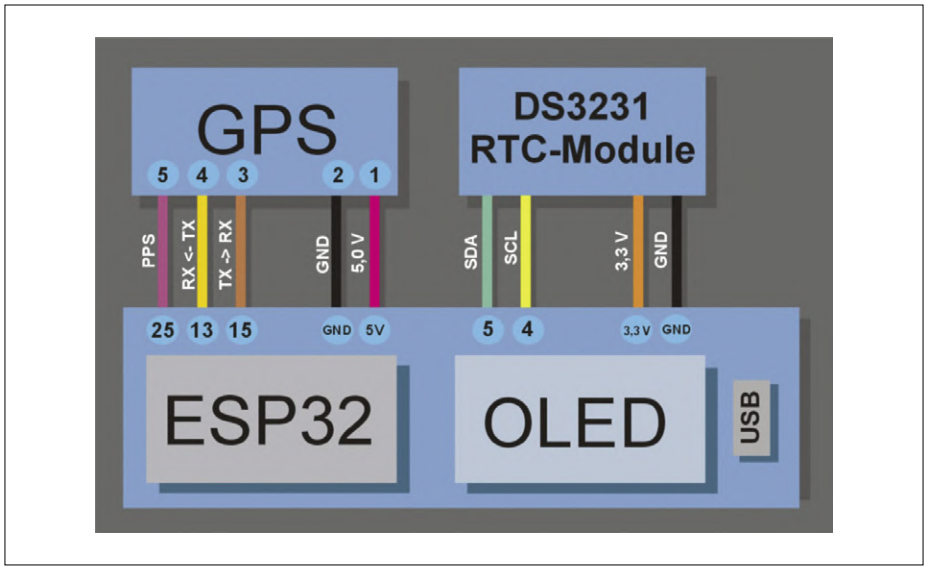


Figure 4. The electrical connections between the three modules.

The Pulse Per Second interrupt

The software configures GPIO25 for the PPS (pulse per second) interrupt, with the internal pull-up resistor enabled in order to ensure that the pin does not take on an undefined voltage level. The interrupt is triggered by a rising edge on the pin, and the function `handlePPSInterrupt` is called. This function must be declared with the `IRAM_ATTR` attribute, which forces it to reside in the RAM of the ESP32. Code not residing in RAM executes more slowly, because of the need to access the external flash memory. Or even worse: if the flash is turned off then an exception will be triggered and the ESP32 will reset itself. The use of floating-point variables in interrupts can also be problematic, as using the floating-point unit in interrupt code in the current version of FreeRTOS can lead to difficulties and unexpected results after the return from the interrupt. The call to `xSemaphoreGiveFromISR` at the end of the interrupt service routine triggers the update of the OLED displays.

An important point to note is that the pins that drive the I²C interface of the RTC from the ESP32 are not the conventional ones: instead, the SCL line to the RTC is connected to GPIO4 and the SDA line is connected to GPIO5. The GPS module uses a normal serial interface, with GPIO13 as the receive (RX) pin and GPIO15 as the transmit (TX) pin. The RX pin of the ESP32 should be connected to the TX pin of the GPS module, and the TX pin of the ESP32 should be connected to the RX pin of the GPS module. Both modules require a 5 V supply: the power and ground connections come directly from the ESP module. The one pulse per second output of the GPS module (pin 5) should be connected to pin 25 on the ESP32 module. **Figure 4** shows the wiring that is required.

In order to see what kind of problems our yet-to-be-written software will have to address, we can first take a look at the freely-available *ESPNTPServer* code [1]. For the 'real' code we will recycle some components from other Elektor projects, in particular the time correction function in the Pinball Clock [2] as well as its ability to synchronize itself with a variety of sources with differing priorities.

For the RTC it is possible to use a DS3231 or a DS1307. After start-up the processor looks for an RTC device at I²C address 0x68: if a response is received then the code assumes that a type DS3231 RTC is connected. If you wish to use the DS1307 you will need to recompile the code with the line in Firmware.ino that says `RTC_DS3231 rtc_clock;` modified to read `RTC_DS1307 rtc_clock;`.

The RTC provides time information when no GPS signal can be received or if the module is not present. The server will also work if the ESP32 cannot find an RTC device: in this mode the ESP32's clock is initialized to midnight at the beginning of January 1, 1970, and will be updated the first time that any GPS data are received. If both RTC and GPS modules are not present then the NTP server will still run, but in this case the time must be manually set and adjusted from time to time. And perhaps it goes without saying that the NTP server is configured using its own web page.

NTP data

NTP uses UDP (user datagram protocol) for the transmission of data packets. The NTP server sits listening on port 123 for an incoming NTP request and

replies with a standard packet of length 56 bytes in the NTPv4 format (described in RFC 5905). The components of the datagram are illustrated in **Figure 5**. *LI*, *VN* and *MODE* are bit fields: *LI* indicates whether a leap second is to be inserted at the end of the current minute; *VN* indicates which version of the protocol is being used; and *MODE* represents the operating mode of the system (in our case, 'server').

Root Delay refers to the distance between this time server and the next. Since ours is the only time server in the (local) world, we set this field to 1. The *Stratum* field is also set to 1 as there is no NTP server between our server and the GPS reference. The *Root Dispersion* field is also set to 1. The possibility of leap seconds is ignored, and for the *Reference Identifier* field the value 'PPS' ('pulse per second') is used, indicating that the server's timekeeping is governed by a generic 1 Hz signal.

Times are reckoned in seconds from the beginning of January 1, 1900, which is 70 years or 2208988800 s before the start of the 'epoch' used as the reference for time in the Unix kernel. Timestamps are represented by two 32-bit fields, one for the number of integer seconds and one for the fractional seconds. The *Reference Timestamp* is intended to hold the current time. Since our server only works in whole seconds, the *Reference Timestamp Fraction* field is always zero, and the same goes for both the *Receive Timestamp* and the *Transmit Timestamp*. The *Precision* field contains the base 2 logarithm of the clock jitter, where the jitter is given as the execution time of the function to read the current time plus one second, which arises as a consequence of our only working in whole seconds. This is not a perfect solution, but is adequate for our purposes.

Time sources

As previously mentioned, the NTP server operates with two external sources of time information. One of these is the RTC, and the other is the GPS and its one-pulse-per-second output. This latter output, which is used to trigger an interrupt, is only used when it is available: under poor reception conditions it can drop out. If the signal is present the system uses the GPS seconds counter; if it is not, an internal counter is used. The one-pulse-per-second signal is monitored using the following algorithm. If no pulse is seen for 1400 ms, then the system assumes that the GPS signal is lost, and switches over to its internal time source. That involves immediately adding one extra second to the time (corresponding to the first missing pulse), and then, 600 ms later (that is, at the point where the next pulse would be expected), adding a further second.

When a GPS signal is received the module outputs time information. It is sufficient to check the GPS time information only once every ten minutes and synchronize the internal clock or external RTC: as long as reception continues, the precise one-pulse-per-second output is adequate to advance the ESP32's internal clock accurately.

For some purposes it is desirable to disable synchronization with absolute GPS time, and use only the one-pulse-per-second output. This can be used to test devices on your local network to verify that they operate correctly in special circumstances such as the changeover between summer time and winter time.

Software

The software takes advantage of a few special features of the ESP32. The updating of the display is done in an infinite loop in a dedicated task, which does not block other processes on the ESP32.

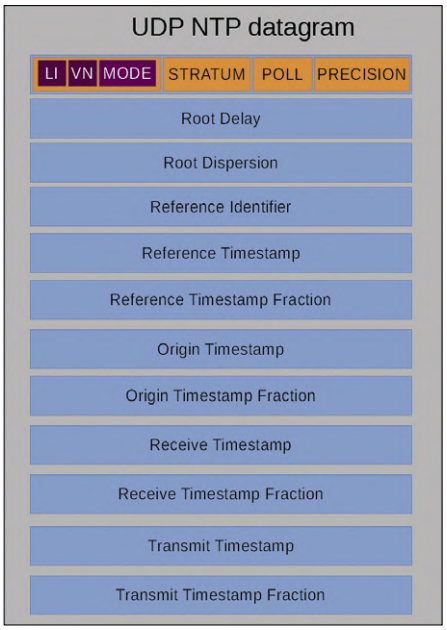


Figure 5. Structure of an NTP datagram.

Inter-process communications are used to tell the task once per second that the contents of the displays needs to be recomputed. The display task remains idle, consuming no processing resources, until a semaphore is activated.

Also, a mutex is used to prevent the main loop or the display task having unfettered access to the I^2C bus. Mutex facilities are provided by the FreeRTOS real-time operating system, which is normally hidden beneath the Arduino framework layer. A description of FreeRTOS, its capabilities and limitations is outside our scope here, but an article on the subject is in the pipeline.

The code to run on the ESP32 is available as a free download from the Elektor web page accompanying this project [3], and any updates will also appear on this page. You can also drop by GitHub [4], where you will find previews of new versions and updates to the original code.

Web Links

[1] ESPNTPServer: https://github.com/liebman/ESPNTPServer

[2] Beat the Elektor Pinball Clock!, ElektorLabs magazine March & April 2019: www.elektormagazine.com/180307-01

[3] Project page accompanying this article: www.elektormagazine.com/180662-01

[4] Firmware on GitHub: https://github.com/ElektorLabs/180662-mini-NTP-ESP32

[5] Arduino setup for ESP32: https://github.com/espressif/arduino-esp32/blob/master/docs/arduino-ide/boards_manager.md

[6] Arduino ESP32 filesystem uploader: https://github.com/me-no-dev/arduino-esp32fs-plugin

[7] Guide to uploading files: https://techtutorialsx.com/2018/08/24/esp32-arduino-spiffs-file-upload-ide-plugin/

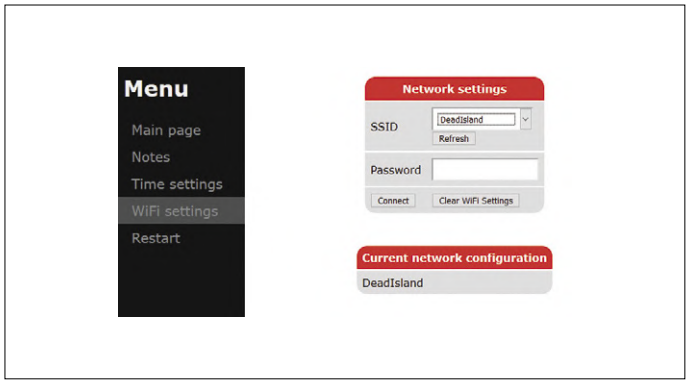


Figure 6. The WLAN settings configuration menu page.

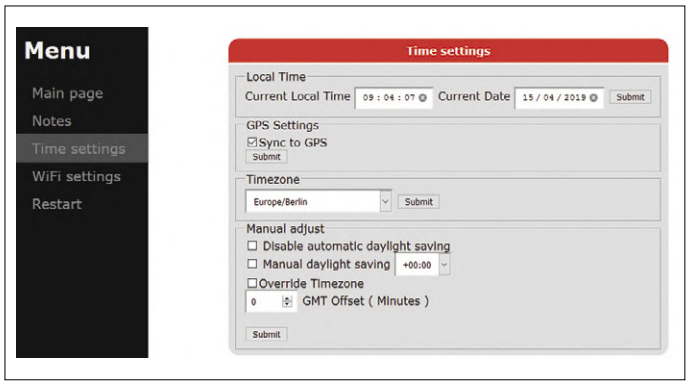


Figure 7. The menu for settings related to the time functions.

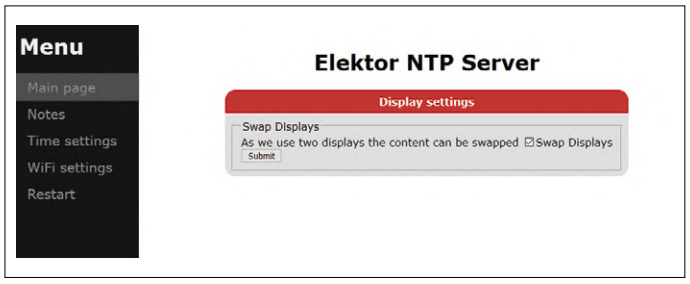


Figure 8. The display configuration menu.

It is a good idea to install the latest versions of the Arduino IDE and the package for the ESP32 board on your PC (see [5]). If you wish to compile the code yourself, you will also need to install the following libraries:

- U8G2
- Time
- Ticker
- TinyGPS++
- RTCLib
- ArduinoJson 6.x
- CRC32

Note that this list might change for newer versions of the project. Once the compiled code has been uploaded to the ESP32 module, it is also necessary to upload the web page it requires. How that is done is described in various places on the Internet and in several Elektor projects that use the ESP32. The Arduino ESP32 filesystem uploader (and documentation) can be found on GitHub at [6], and a guide to how to upload files (a web page is just a file) can be found at [7]. When the device is reset the NTP server will be ready for use as soon as the boot process is complete.

Configuring the server

Configuring the server is very straightforward. Once the code and web page have been successfully uploaded the server starts up as an access point. In this mode it is possible to connect to the device and access its web page at the local IP address 192.168.4.1. There you can enter the SSID and password for your WLAN (see **Figure 6**). The web interface can also be used to configure settings related to the time functions (**Figure 7**). Once the WLAN settings are activated, the server attempts to connect to the WLAN and obtain a local IP address using a DHCP request.

Finally, the web interface can be used to adjust other configuration settings (see **Figure 8**). Under the *Main Page* menu item it is possible to exchange the contents of the two displays. If only one display is connected, this option will select what is shown on it. And that completes the configuration of the mini NTP server. From now on it should happily serve up time information in response to incoming NTP requests. ⏮

(180662-02)

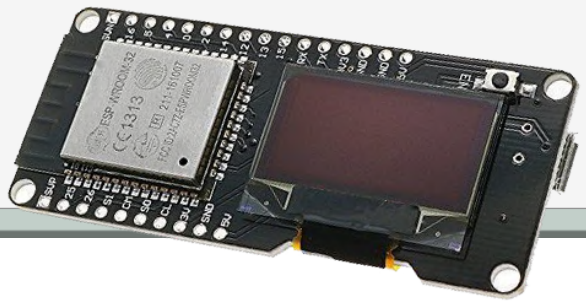
@ WWW.ELEKTOR.COM

→ OPEN SMART GPS
www.elektor.com/open-smart-gps-serial-gps-module-for-arduino-apm2-5-flight-control

→ Lolin ESP32 with OLED
www.elektor.com/wemos-lolin-esp32-oled-module-with-wifi

→ Adalogger FeatherWing: RTC + SD add-on
www.elektor.com/adalogger-featherwing-rtc-sd-add-on

→ 0.96 inch I²C OLED display (optional)
www.elektor.com/blue-0-96-oled-display-i2c-4-pin



productronica
fast forward
the start-up platform
powered by Elektor

COMPETE TO LAUNCH YOUR STARTUP @ PRODUCTRONICA 2019

p-ffwd 2019 – Participate Now!

November 12-15, 2019
Munich



Further information at:
www.elektormagazine.com/p-ffwd

Platinum Sponsor:
DISTRELEC
Distribution with a difference

Productronica Fast Forward is brought to you by
productronica
elektor
INNOVATION STARTUP TRADE

Creating a New Component (Symbol) in KiCad

and preparing to associate an existing footprint

By **Dr. Peter Dalmaris** (Australia)



In this "recipe" you will learn how to create a custom component (or symbol as it is also known). You use symbols in schematic diagrams that you can create in Eeschema. Much of what you will learn in this recipe you will be able to reuse for modifying existing symbols. There is a separate recipe in the book that explains how to do that.

Editor's note: This article presents a sample chapter from the book "Kicad Like a Pro" by Dr Peter Dalmaris, reformatted to match *ElektorLabs* magazine page layout. The book is available from the Elektor Store.

Most likely, you want to create a custom schematic symbol because you have a physical component but can't find an existing symbol to represent it. Perhaps you have searched through the symbols that come with KiCad, and Googled for suitable third-party symbols, but couldn't find any.
It is also likely that you will want to associate your new symbol with a footprint. If your physical component has a standard package, like DIP, for example, then you will be able to associate an existing footprint with your custom symbol. If not, you will also need to create a custom footprint. You can learn how to do this in the relevant recipe, also available in this book.
In this recipe, you will learn how to create a custom symbol by creating a symbol for the 555 timer integrated circuit. There are plenty of libraries for KiCad that contain this symbol, but for the sake of learning, let's pretend that we can't find it.
What we want to create is a symbol like the one in **Figure 35.1**. Our objective is to create a symbol that complies with the convention. In regard to IC symbols:

- We arrange pins around a rectangle.
- We group pins according to function (like inputs, outputs, power etc.).
- We place the Vcc pin on the top of the rectangle.
- We place the GND pin on the bottom of the rectangle.
- We choose an appropriate name and designator for the symbol. Symbol designators are standardized; you can learn more about them at [1].

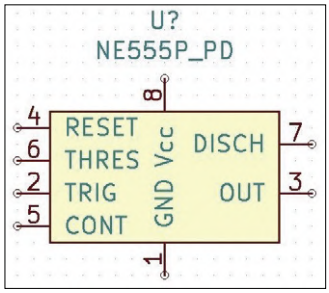


Figure 35.1: A custom-made symbol for the 555 IC.

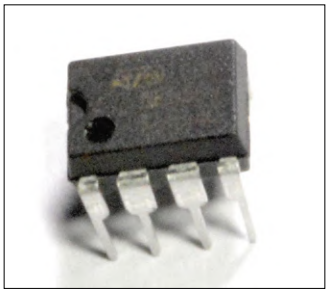


Figure 35.2: The physical component is a type 555 timer in a DIP package with 8 pins.

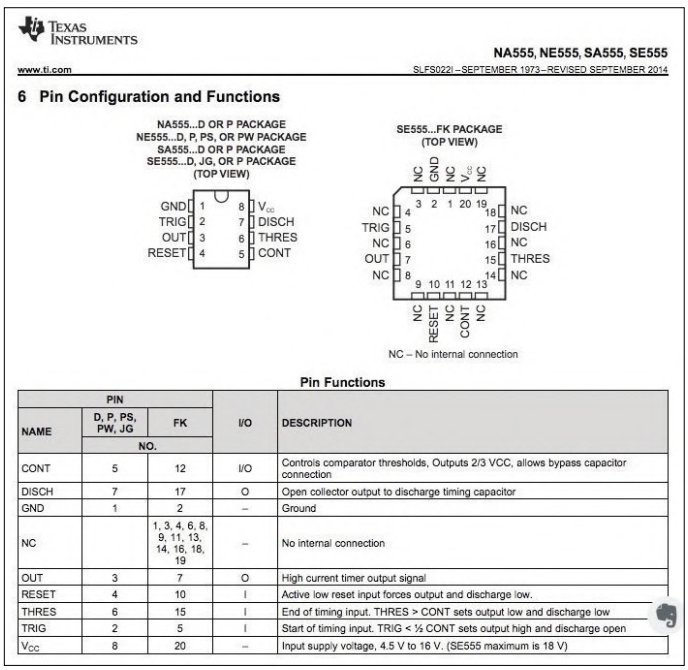


Figure 35.3: Pin configuration and functions from the IC's datasheet.

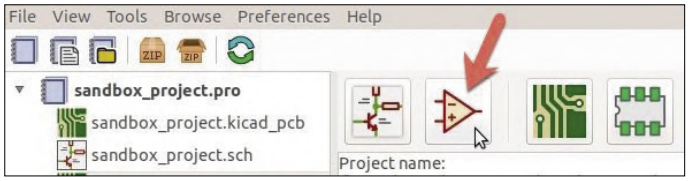


Figure 35.4: Start the Symbol Editor.

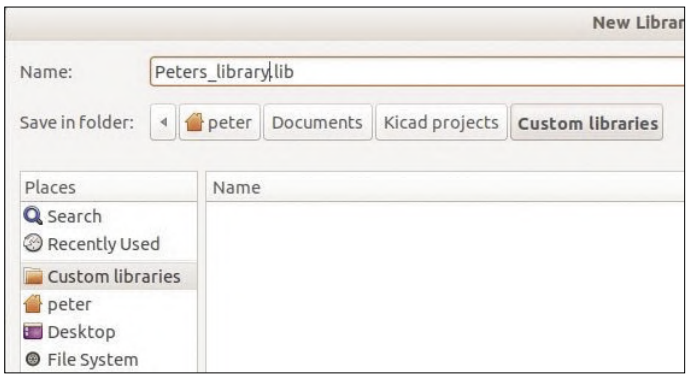


Figure 35.5a: I have placed mine in a directory named 'Custom libraries'.

The physical component that we are working with is in **Figure 35.2**. This component comes in a standard DIP package with 8 pins.
Because we are working to create a symbol, we don't need to know anything about the physical characteristics of the physical component other than the total number of pins that its package contains. You need to know details about the physical characteristics of the component when you are working on its footprint.
However, it is beneficial if you have access to the components data sheet. The data sheet contains information that you need: The names, numbers, and roles of each pin, whether they are input, output, bidirectional, power, signals, etc. All this is useful information, and the more you have on hand, the better.
The datasheet for the example component is available from its manufacturer. The information you need is on page 6 in the book, and I have included it in **Figure 35.3** for your convenience.

Let's start the process of creating a new symbol. In the main KiCad window, click on the Symbol Library Editor button (**Figure 35.4**).
You must store each symbol inside a library file, so before you start creating the symbol create a new library. Create a new library by clicking on the 'New library' button, or through the File menu (**Figure 35.5a**).

KiCad will ask you if you would like this library to be available to all projects ('Global') or only to the current project ('Project'). Choose the most appropriate one for your circumstances (I chose 'Global'). You are now working in the new library, and you will store the new symbol in it. You can confirm this by looking at the header of the Editor window. The path and name of the library you just created must be showing there.
Next, click on the 'Create new symbol' button in the top toolbar. A prompt will ask you to select the library in which you will store the symbol. The library you just created should be listed. Click on it to select it and click 'OK' to continue. The Symbol Properties window will show up. The most important values that you need to complete are the symbol name and designator. The name typically consists of the physical component's model name and any other information that helps to identify it. When you use it later, you have to search for it in the symbol library and having a good name will help you find it quickly. In my example, because I want to differentiate my 555 symbol to those that exist in other libraries, I add my initials 'PD' at the end of the name.

For the designator, you should not guess. Visit Wikipedia to see the Reference Designators table. You can see a section of this table in **Figure 35.5b**. The designator for integrated circuits is 'U,' so type this in the Default Reference Designator field.

Figure 35.6 shows the values I have entered in the Symbol Properties window. Other than the symbol name and the designator, everything else remains as per the default.
Click 'OK' to commit and continue. KiCad will place the designator and symbol name in the middle of the sheet, on top of the other. Use the 'M' hotkey to relocate the two blocks of text. You should have something similar to the example in **Figure 35.7**. Continue by drawing the outline of the symbol. You can either use the polygon tool or the rectangle tool to do this. Your outline should like the example in **Figure 35.8.**

Add the background colour that is consistent with other IC symbols by opening the drawing properties window for the rectangle (place your mouse pointer on the rectan-

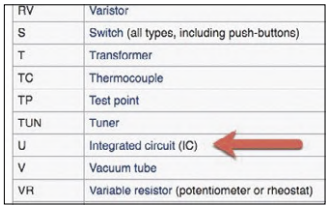


Figure 35.5b: A section of the reference designators standard IEEE 200-1975/ANSI Y32.16-1975.

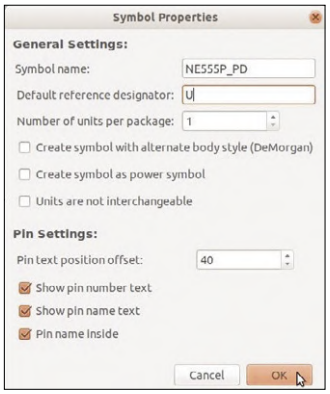


Figure 35.6: The properties for the new symbol.

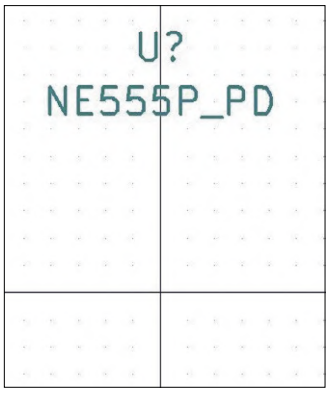


Figure 35.7: An empty new symbol.

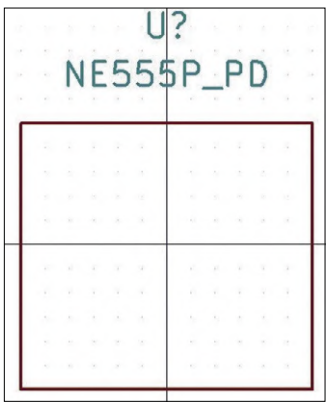


Figure 35.8: The outline of the footprint.

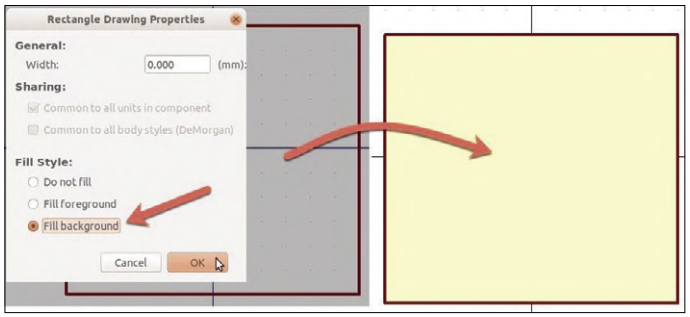


Figure 35.9: Fill the rectangle background.

gle line and type 'E'). Under 'Fill Style', check the 'Fill background' radio button (**Figure 35.9**).

Next, work on the pins. Keep the datasheet open because you need the information in it. For your convenience, refer to the excerpt in Figure 35.3. Click on the pin button in the right toolbar. Place the 8 pins around the perimeter of the symbol outline, as in the example of Figure 35.1. Important to remember here is that the convention is to group similar pins together and place the two power pins to the top and bottom of the rectangle. Let's start with the Vcc pin. According to the datasheet, the Vcc pin is number 8, and according to the convention, it should go to the top edge of the rectangle. Click on the pin tool, and then click in the middle of the top edge. The Pin Properties window will come up. Fill it as shown in the example in **Figure 35.10**.

The fields inside the box are the ones that you need to focus on. The pin name is arbitrary, but of course, you should use a name that is appropriate. I usually use the same name that I see in the documentation for this pin. The Pin number, on the other hand, is very important. The Pin number is how schematic symbols and footprints can associate physical and symbolic pins. When you design the custom footprint for this physical component in the 'Creating new footprint' recipe, it is the number that you put in the Pin number field that dictates which net in the schematic diagram connects to the correct pad in the footprint. Take the pin number for the Vcc pin from the documentation ('8') and type it in this field.

In the orientation drop-down, select the option that matches the side of the rectangle to which you are attaching the pin. The Vcc pin should go on the top of the rectangle and should have its circular connector pointing away from the rectangle. The horizontal line of the pin orientation icon represents the rectangle. If you wanted to place the pin on the left of the rectangle, you would choose the icon with the circular connector pointing towards the left.
Finally, because the Vcc pin is a power pin, I have selected the 'Power input' electrical type. You should choose the same type for the GND pin.
Click 'OK' to commit the changes. Place the pin in the middle of the top side of the rectangle, as you can see in **Figure 35.11**.

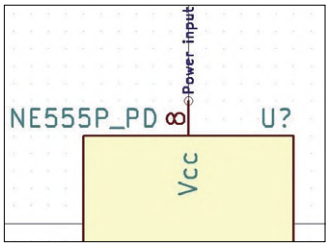


Figure 35.11: The Vcc pin, in place.

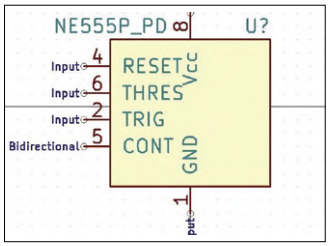


Figure 35.12: Input pins are placed on the left side.

I have moved the text block so that it doesn't overlap with the pin.
Follow the same process to add the GND pin (pin 1) in the bottom edge of the rectangle. Copy the pin name and number from the datasheet and mark it also as a Power Input.
Continue with the left side of the rectangle where you should place the input pins. According to the datasheet, the input pins are RESET, THRES, TRIG. There is one bidirectional pin, 'CONT,' which you can place either on the left or the right of the rectangle. I have placed it on the left. Your symbol should look like the example in **Figure 35.12**.

The pin attributes for the input pins on the left of the symbol look like the example in **Figure 35.13**. Remember that pin 5 is bidirectional so its electrical type should be 'bidirectional'.
Continue in the same way to create the last two pins. Those are output pins, as per the datasheet. Use the pin names and

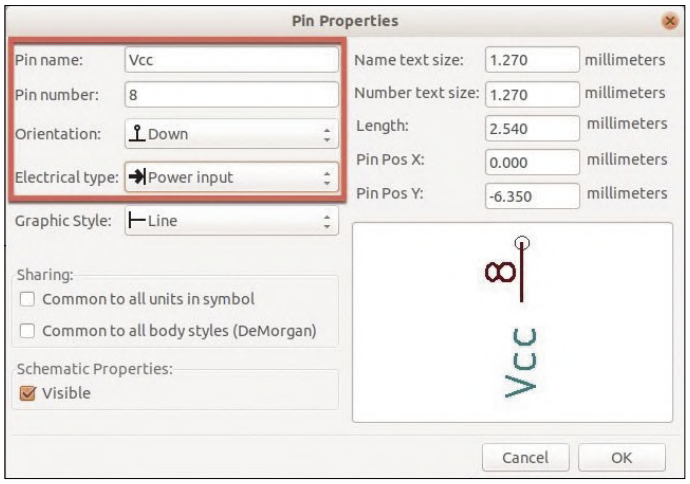


Figure 35.10: The Vcc pin properties.

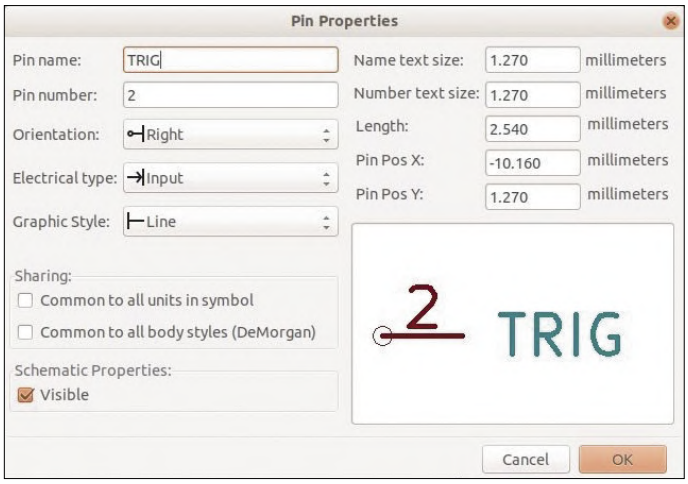


Figure 35.13: The input pin attributes.

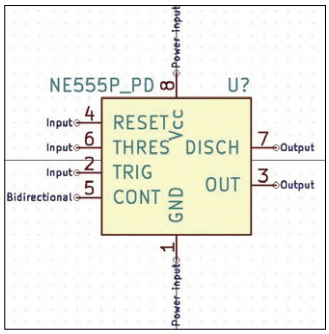


Figure 35.14: The completed custom symbol.

numbers as they appear in Figure 35.3 for pins 7 and 3. When completed, you should have a symbol that looks like the example in **Figure 35.14**.

The last thing to do is to add the URL for the real-world component datasheet to the symbol properties. This will be useful to you for future reference. It will be surely useful when you go ahead to create a custom footprint to match the symbol. To add the datasheet URL, click on the Symbol, 'Fields…' to bring up the Field Properties window. Click on the Datasheet row to select it and then copy/paste the URL in the Field Value field (**Figure 35.15**). Click OK to commit the changes.

Your work is complete. Save the symbol to the selected library by clicking on the Save Current Symbol button. Then, test that you can use it in Eeschema. Open Eeschema. Go into Symbol Libraries from the Preferences menu and add the new library (read the relevant recipe if you don't know how to do this). Place your cursor on the Sheet and type 'A' to add a new symbol. Search for the name of your library by typing part of its name in the filter field (**Figure 35.16**). The library should appear. Double-click on the symbol to drop it on the sheet.

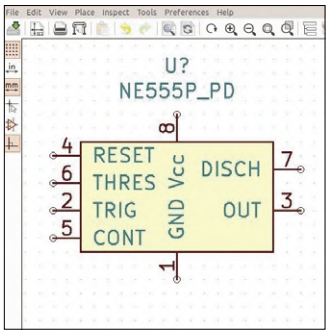


Figure 35.17: Your new custom symbol in Eeschema.

The custom 555 symbol should now be in place inside Eeschema, and you can go ahead to use it as you do with any other symbol (**Figure 35.17**).

In this recipe, you learnt how to create a brand-new symbol. What if you have found a symbol that is close to what you want, but could be perfect with a bit of tweaking? In other words, what if you want to modify an existing symbol? You can learn how to do this in the '36. Modifying an existing component (symbol)' recipe. ⏮

(190041-01)

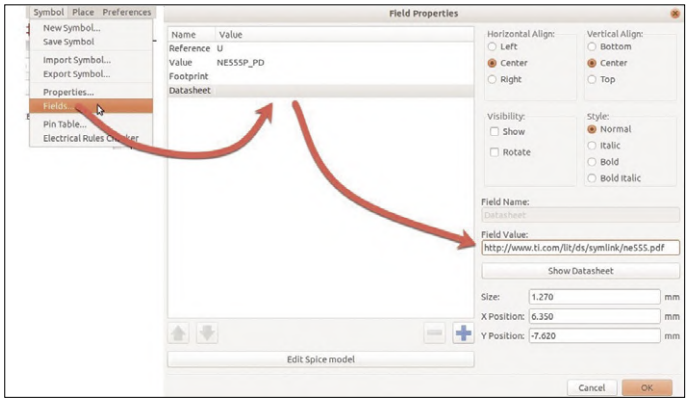


Figure 35.15: The datasheet is very useful to have readily available.

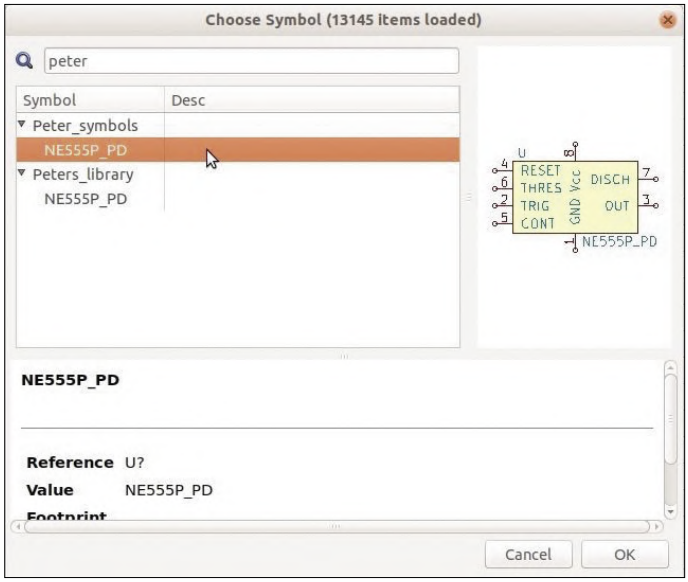
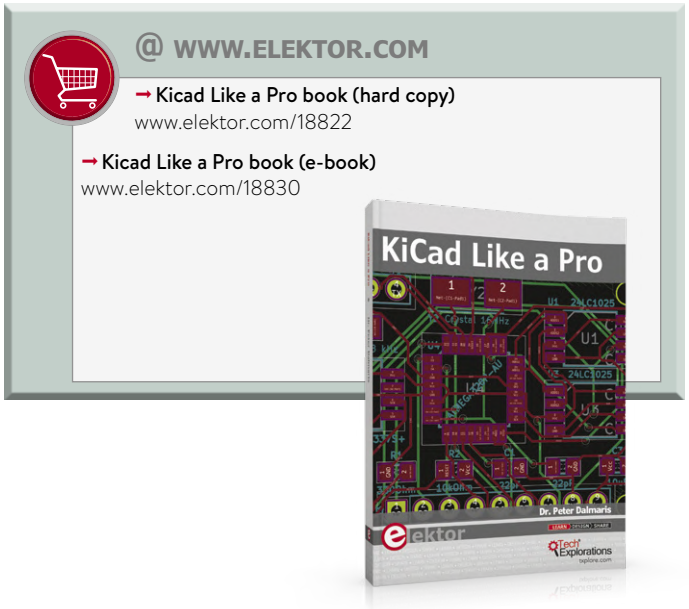


Figure 35.16: Find your new library and symbol.

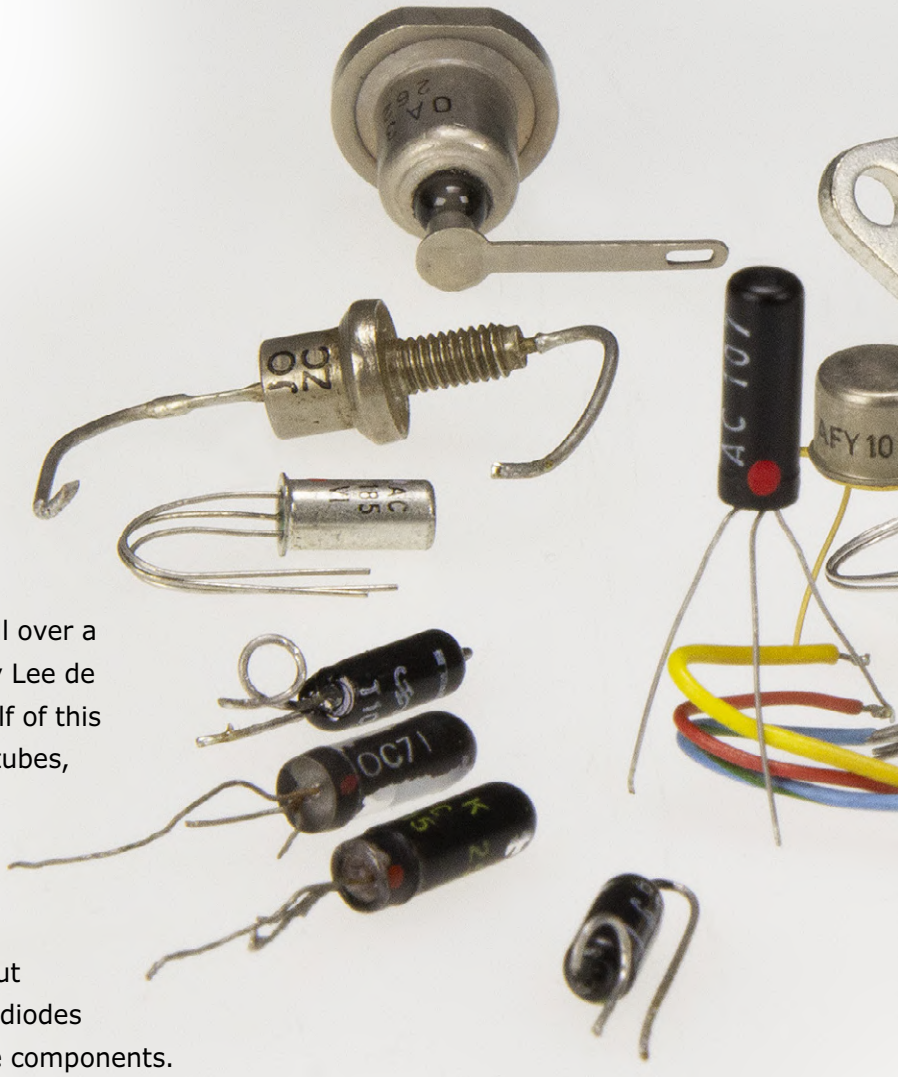


Web Link

[1] Symbol designators: https://en.wikipedia.org/wiki/Reference_designator

From Tubes to Silicon

With an interesting intermezzo



By Dr Martin Beusekamp, MSc

We have been living in the electronic age for well over a century now, since the invention of the triode by Lee de Forest in 1907. Roughly speaking, in the first half of this period electronics was based solely on electron tubes, and in the second half almost entirely on silicon semiconductors in discrete or integrated form. In a relatively short transition period, electronic equipment was designed and built with neither tubes nor silicon semiconductors, but instead with selenium rectifiers and germanium diodes and transistors. Let's have a closer look at these components.

Selenium rectifiers

Figure 1 shows a typical selenium rectifier, consisting of a number of small metal plates made of steel or aluminium and coated with a thin layer (approximately 1 µm) of bismuth (Bi) or nickel (Ni), with a thicker layer (approximately 60 µm) of selenium (Se) on top. This sandwich structure acts as a diode because electrons can cross the barrier between the selenium and the steel or aluminium more easily in the one direction than in the other direction.

Each of these diodes has a reverse blocking voltage of approximately 30 to 50 V, but to be on the safe side the generally accepted rule was a maximum of 20 V per diode. The overall rectifier in Figure 1 consists of eight plates, so it has a maximum reverse blocking voltage of 160 V. The maximum current through the rectifier is linearly proportional to the surface area of the metal plates, and in practice it is approximately 25 to 30 mA/cm^2 [1]. This means you don't need to look for a part number to determine the properties of a selenium rectifier — just count the number of plates for the blocking voltage and measure the size of the plates for the forward current.

Due to the relatively high forward current capacity of selenium rectifiers with large dimensions, for a short while they were used to rectify currents up to several amperes. However, users often said that with selenium rectifiers the question was not how long they would last, but instead when they would fail. Their reliability and lifetime were in fact fairly limited, certainly compared to the indestructible silicon diodes we use now (provided that the voltage, current and power remain within the specified limits).

Selenium rectifiers generally fail due to

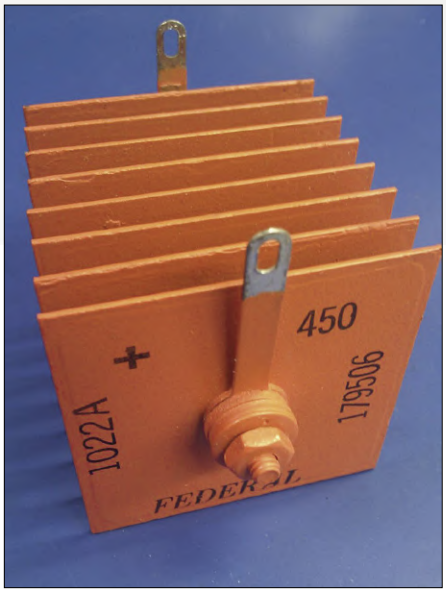
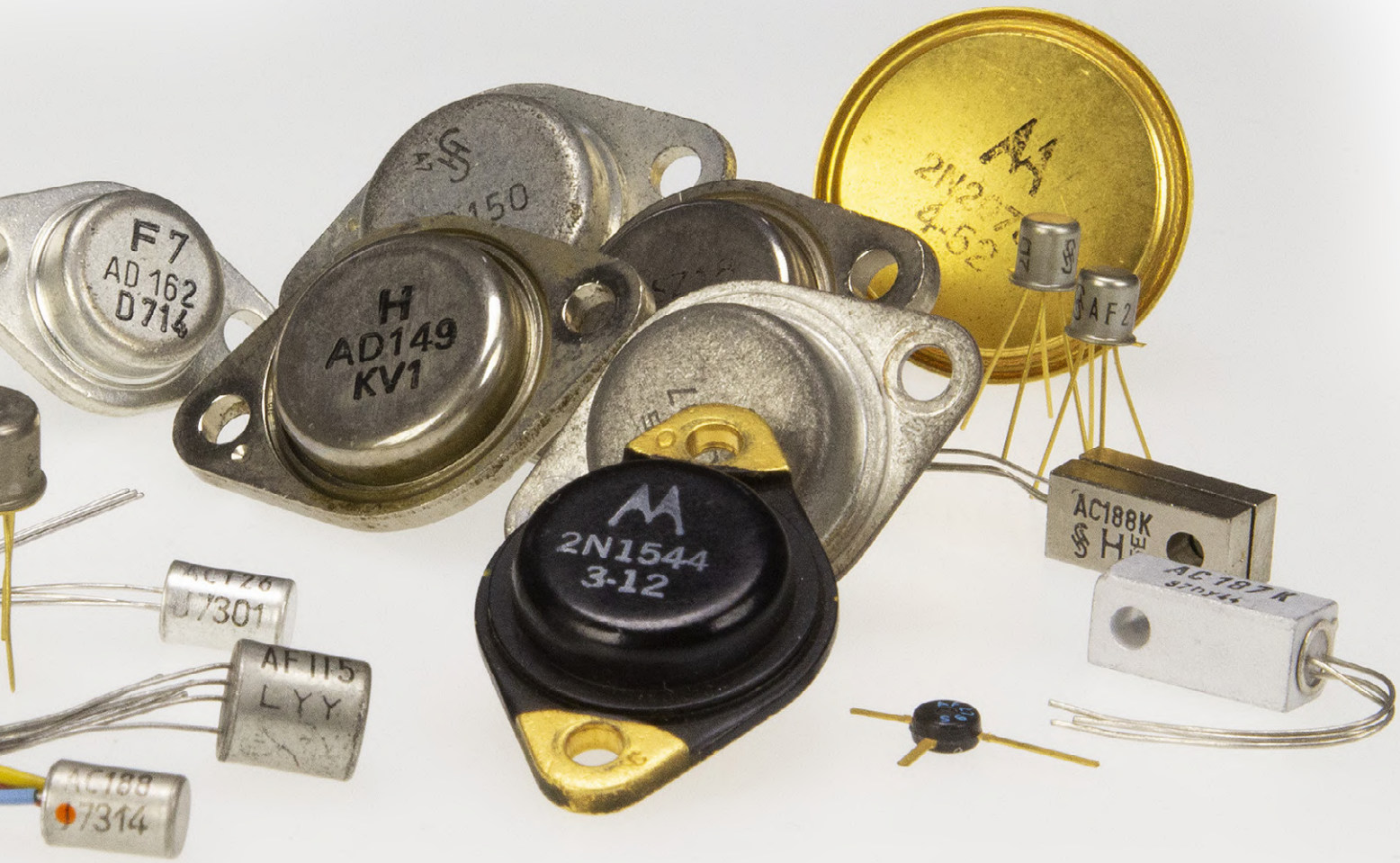


Figure 1. A typical example of a selenium rectifier. (Creative Commons Attribution-Share Alike 3.0 Unported)



ESTD 2004

www.elektor.tv

Retronics is a regular section covering vintage electronics including legendary Elektor designs.
Contributions, suggestions and requests are welcome; please telegraph editor@elektor.com

overheating as a result of increased resistance in the forward conduction direction. This is accompanied by a penetrating odour and the release of a gas that can certainly be hazardous in relatively high concentrations (see the inset '**Hold your nose, open the window, and get out**'). If you encounter a selenium rectifier that is still working, you are well advised to replace it by a silicon diode with a series resistor to obtain the same voltage drop as the original selenium rectifier.

Germanium diodes and transistors

The first semiconductor devices that became available on a large scale were germanium diodes and transistors (see the inset **'Tube codes for diodes and transistors'**). Germanium (Ge) is an element that is located directly below sili-

Hold your nose, open the window, and get out.

As mentioned in this article, selenium rectifiers can sometimes be dangerous. The internal resistance of selenium rectifiers rises as they get older, which increases the voltage drop over the rectifier. This in turn increases the power dissipation, so the rectifier gets warmer. At some point it will be so hot that the selenium spontaneously starts to oxidise and turn into selenium dioxide (SeO_2).
Selenium dioxide is a solid substance at room temperature, but it escapes from an overheated selenium rectifier as a greenish gas. This gas is toxic – in fact, highly toxic. It's bad enough that it can make you dizzy, nauseous and tired, but what even worse is that it can cause permanent damage to your liver and spleen.
Fortunately, people have a natural defence mechanism against selenium dioxide. Most people can smell it at concentrations as low as 200 ng/m³, so they are alerted before the situation becomes truly dangerous. The smell of selenium dioxide is also so disgusting that everyone wants to get away from the source.
Selenium is located directly below sulphur (S) in the periodic system. This means that selenium and sulphur have comparable properties, the same way as germanium and silicon, which are also vertically adjacent as described in this article. That's why the odour of selenium dioxide resembles the odour of sulphur dioxide. It's like rotten onions with garlic.
See also Barry L. Ornitz, *Selenium Rectifier Replacement* [7], and 'Sleepless nights from a pile of onions' [8].

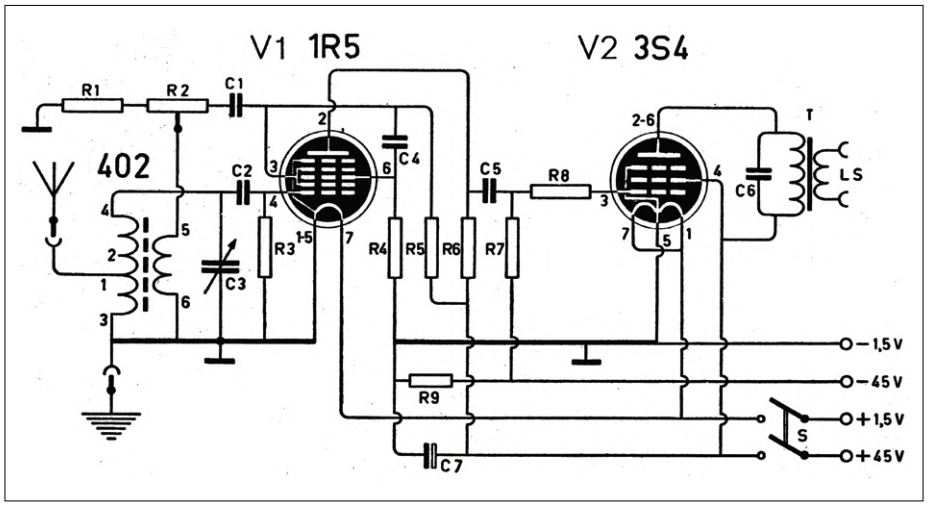


Figure 2. Schematic diagram of the Nucleon construction kit from Amroh, with two tubes and two batteries for power supply.



Figure 3. Cover page of the July 1960 issue of the Dutch youth magazine Radio Blan. Note the price (in cents of a Dutch guilder, of course).

con (Si) in the periodic table. This means that germanium also has four valence electrons, resulting in an orderly spatial crystal structure with the pure form of the substance.

Like silicon, germanium can be doped with an element having one more electron in the outermost shell, such as phosphorus (P) or arsenic (As), to produce n-type germanium with an excess of free electrons. Doping with an element having one electron less in the outermost shell, such as boron (B), results in p-type germanium with a shortage of free electrons. Germanium diodes and transistors have the same structure and operating principle as their silicon cousins, and they look approximately the same (see the lead photo). However, they have different electrical characteristics due to the differences in the material properties.

For example, the voltage drop over a conducting germanium diode or a conducting base-emitter junction of a germanium transistor is only 0.1 to 0.2 V, compared to 0.6 to 0.7 V with silicon types. This advantage is offset by a variety of disadvantages, such as the higher internal resistance of a germanium diode and the higher leakage current in the blocking direction – on the order of 1 µA with germanium, compared to 1 pA with silicon. It's also not possible to fully cut off a germanium transistor. Even without a base current, there is a collector current ICEO of several microamps with small-signal germanium transistors or several milliamps with germanium power transistors.

A wide variety of other problems, such as temperature dependence, changing characteristics with ageing and the tolerance range in mass production, were also significantly greater with germanium semiconductors than with their silicon replacements later on (yet another problem is described in the inset **'Just a whisker away'**). Once the technology for the production of silicon semiconductors was sufficiently advanced, germanium semiconductors disappeared fairly quickly with the exception of a few applications where their low forward voltage was essential.

Tube codes for diodes and transistors

The first germanium diodes, with type numbers such as OA79 and OA81, came onto the market in the mid-1950s. A few years later they were joined by the first germanium transistors, with type numbers OC3, OC4, OC13 and OC14.

These type numbers were based on the usual European codes for electron tubes at that time. The 'O' stands for a cold cathode (no filament present), the 'A' stands for a single diode, and the 'C' stands for a triode (the simplest type of gain element).

Specific European codes for semiconductor devices were agreed and introduced only later, with the first letter 'A' for all materials with a band gap of 0.6 to 1.0 eV (which in practice is always germanium), 'B' for all materials with a band gap of 1.0 to 1.3 eV (in practice always silicon), and 'C' for all materials with a larger band gap (such as gallium arsenide). They form the basis for type numbers such as CQY37 for a particular infrared LED or CQY80 for a particular optocoupler.

If you have a diode with an OA type number or a transistor with an OC type number, it was probably made using a glass tube coated with black paint. If you scratch off the paint, you have a photodiode or phototransistor.

Construction kits for youngsters

Even before the invention of germanium semiconductors there were many DIY kits with electron tubes, such as the Philips 'Senior' kits for a variety of medium-wave receivers and amplifiers. These DIY projects were only suitable for adults, due to the high voltages in the circuits and the need for soldering.

The manufacturer Amroh tried to do something about this by launching some kits with electron tubes designed for use with batteries. The Amroh Nucleon medium-wave receiver (**Figure 2**) with loudspeaker output, for example, used two tubes: a DK91 (1R5) and a DL92

(3S4), with filament currents of 50 mA and 100 mA, respectively, at 1.4 V. A good-quality LR20 battery (also known as a D cell) could keep the filaments glowing for several hours, until the terminal voltage dropped to 1.2 V or less and operation of the radio was reduced to a minimum. The anode voltage of the Nucleon was provided by a 45 V anode battery (B battery). Until the end of the 1950s, B batteries with high voltages (up to 90 V) were readily available for anyone who could afford them.

And then came the germanium diodes, with which even relatively young kids could build a crystal receiver such as the Pioneer I from Philips or the 'jam jar receiver' from the magazine Radio Blan for young hobbyists (**Figure 3**). Note the price of the magazine: 15 cents (of a Dutch guilder). Ignoring correction for inflation, that's equivalent to a bit less than €0.07, £0.06 or $0.08. This was low cost even at that time (1960), but the magazine was sponsored by dozens of component dealers who sensed a new market.

Germanium diodes were soon followed by germanium transistors, which could be used to build more selective medium-wave receivers with crystal headphones, such as the Philips Pionier II with two OC13 transistors (**Figure 4**), or even loudspeaker output, such as the Philips Pionier III with an additional OC14 as an output amplifier.

A short time later, Philips came out with the EE8 and EE20 experimenter kits (EE stands for 'electronic engineer', and the number stands for how many different devices could be made with the kit in question). The germanium transistors in these kits were a bit more advanced: AF116 and AC126. A wide range of devices could be built, including a phono preamplifier, electronic organ, intercom, Morse code trainer, flashing light, acoustic relay, moisture indicator, timer switch, Wheatstone bridge, and of course various small medium-wave radio receivers (when there was still something worth listening to on that band).

Best of three worlds

Of course, selenium and germanium components were also used in industrial applications where they could make an instrument better, smaller, cheaper, and/or more reliable. However, silicon components became available before germanium transistors could have replaced all electron tubes.

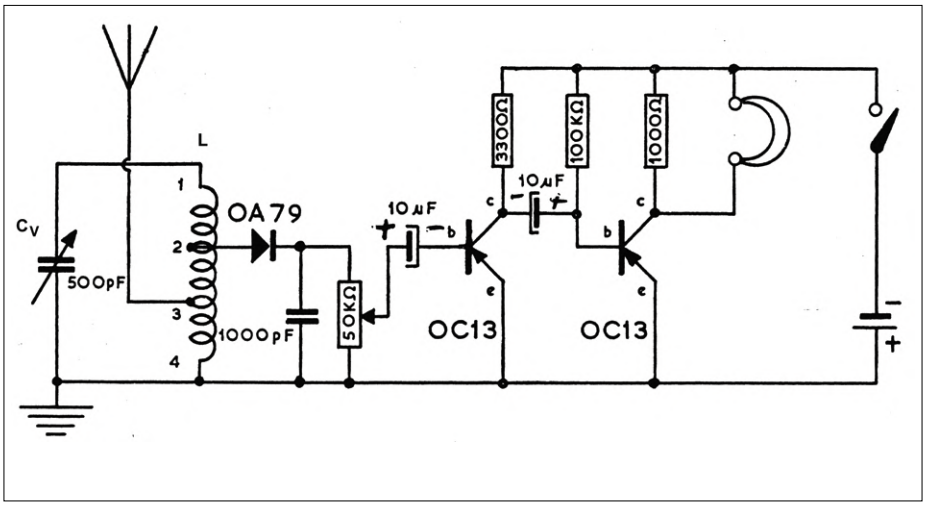


Figure 4. Schematic diagram from the Philips Pionier II construction kit, with a type OA79 germanium diode and two type OC13 germanium transistors.

The Philips PM3230 oscilloscope shown in **Figure 5** is a typical example of what could be called 'the best of three worlds' at that time. Along with the cathode ray tube, this instrument contains seven other electron tubes (five ECC88 and two E180F). In addition, this oscilloscope contains 81 semiconductor components, of which 17 are germanium (primarily



Figure 5. The Philips PM3230 oscilloscope with electron tubes, germanium transistors, silicon diodes and silicon transistors in its circuitry. Photo: Gerrit Busscher.

Boiled or baked?

Semiconductor devices are sensitive to overly high temperatures. This means they need to be soldered quickly. Germanium semiconductors are distinctly more sensitive to overheating during soldering than silicon semiconductors. Accordingly, there were various means commonly used to avoid damage to germanium components during PCB assembly. A loose crocodile clip attached to the lead to be soldered was usually sufficient to dissipate the heat.

The magazine *Radio Blan* (see Figure 3) realised that youngsters might not have much soldering experience and also had to save all their pennies for a single diode or transistor. Corrected for inflation and converted to euros, the price of a germanium diode in 1960 was approximately three euros (£2.50, $3.35), and a germanium transistor cost around twenty euros (£17.25, $25.00).

The tip from 'Uncle Blan', the magazine's guru for everything to do with electronics, to avoid overheating was accordingly: "*Ask your mother for a slice of raw potato. Cut it halfway through, and then place it on the lead of the diode or transistor to be soldered. After soldering, carefully remove the potato slice.*"



Figure 6. The Philips IC2000 medium-wave radio (1976). The compact round radio fits in the case; the device on its right is the mains power supply and charger.

AV11x high-frequency transistors) and 64 are silicon (mainly diodes and bridge rectifiers).

Why did the Philips designers and developers choose these components? The answer is that an analog oscilloscope operates with a wide variety of electronic circuits. The input amplifiers must process signal voltages in the millivolt range, while the post-acceleration voltage of the cathode ray tube is many kilovolts. The power supply generates DC voltages, while the input signals can have frequencies up to 10 MHz with the PM3230, or an order of magnitude higher with more sophisticated (and more expensive) oscilloscopes available then. At that time, some of these functions could still be best implemented with electron tubes, while germanium transistors or even silicon transistors were a better choice for other functions. If you're interested, you can find many more details in the service manual for the PM3230 [2].

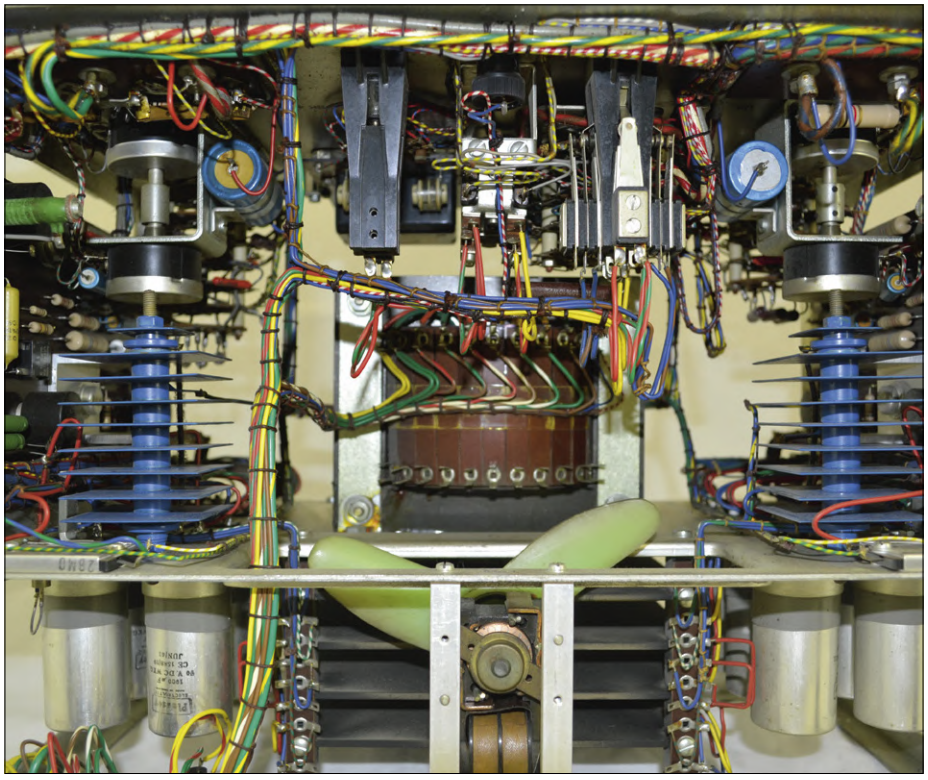


Figure 7. Internals of a Philips type PE4804 power supply, with exclusively selenium rectifiers and germanium semiconductors. Gift from E.G. Pleiter (Hengelo). Photo: Rob Quentemeijer.

In some cases, marketing considerations were behind the use of several sorts of active components. At the IFA in Berlin in 1967, Philips presented a medium-wave receiver with a futuristic appearance, the IC2000 (see **Figure 6** for an example in remarkably good condition).

Approximately three-quarters of the round enclosure of this radio, with a diameter of 7 cm and a height of 3 cm, is taken up by the loudspeaker. A round PCB with two ICs, among other components, is mounted on the loudspeaker magnet. One of these ICs (a TAA263) contains no less than three transistors. The radio also uses one discrete silicon transistor, one silicon diode, and two germanium transistors in the output stage.

The use of ICs in this relatively simple small receiver is certainly not necessary, but it fits with the external appearance and the chosen model number IC2000. At that time, 'IC' was a sort of magic word, and in the latter decades of the previous century the number 2000 was always associated with the technological future wherever it appeared. At that time there were also other manufacturers who tried to boost their sales figures by using 'IC' and '2000' in their model numbers.

If you would like to know more details about this radio, the best place to look for the service manual [3] is the archives of the Dutch association for the history of radio, NVHR (Nederlandse Vereniging voor de Historie van de Radio).

Best of one world

The situation with a relatively simple device such as an industrial DC power supply (non-switching) is completely different. No high frequencies, no very low or very high voltages, a straightforward design, and no need to attract the general public. That's why only selenium and germanium components were used in

Just a whisker away

The first car and the first train were far from being as reliable as the best horse-drawn carriages at the time. The car pioneer and industrialist Henry Ford is credited with saying, "If I had asked my customers what they wanted, they would have said faster horses."
At the end of the tube era, there was a similar situation with transistors (in particular germanium transistors). They were a lot smaller and more energy efficient than electron tubes, but certainly not as reliable.
Although the manufacturers of early germanium transistors (OC170–OC171, AF114–AF117, etc.) made good products, they were not in a position to know what the lifetime of their new marvels would be in practice. It turned out that the lifetime was limited by the spontaneous growth of metal whiskers on the inside of the tin packages of these transistors. As it happens, tin is one of the metals (along with antimony, cadmium, indium and zinc) that is prone to the formation of whiskers on the surface of the material (see **Figure 8** and the Wikipedia description [9]).

The tin packages of early germanium transistors were hollow. Although the transistor was embedded in a blob of silicone grease, there was usually an air space in the package. Spontaneously growing metal whiskers, a few micrometres thick and protruding into the air space, could in time cause short circuits between the transistor package and the electrodes of the transistor (collector, base, and/or emitter).
This problem occupied the attention of all manufacturers of electronic devices during the 1950s and 1960s, and it was even the subject of a specific investigation by NASA. The problem was fully resolved with the advent of transistors in plastic packages or in metal packages made of an alloy that is not prone to the formation of metal whiskers.
See also 'The OC171 Mystery' [10] and the article by Mark Hennessy [11].

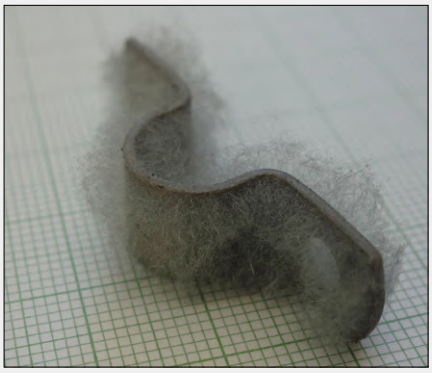


Figure 8. Formation of metal whiskers on a zinc bracket (source: Wikipedia).

the Philips DC power supplies PE4802 (0–15 V / 6 A) and PE4804 (2 × 0–30 V / 2 × 2 A) produced in the early 1960s. **Figure 7** shows the internals of a PE4804 from the Historical Study collection of the faculty of Electrical Engineering, Mathematics and Informatics of the University of Twente [4], which also includes a PE4802. The large blue components are of course the selenium rectifiers, and from the symmetry of the left and right sides in the photo you can clearly see that this is a dual power supply.
The black heat sinks of the germanium power transistors are clearly visible below the light green blades of the fan. By the way, the motor that drives the fan blades is a standard synchronous motor operating from the AC power line, which was used in all Philips record players at that time. More details about this device can also be found in the service manuals [5],[6] on the NVHR website.
These Philips PE4802 and PE 4804 power supplies are collector's items — not because they are especially high-tech, but from the historical perspective: no tubes, no silicon. ⏮

(180574-D-03)

Links and literature

[1] Source: E.A. Harty – Characteristics and applications of Selenium-Rectifier Cells – Transactions on Electrical Engineering, volume 62, October 1943.

[2] Philips PM3230 oscilloscope (Dutch): www.radiodatabase.nl/meetapparatuur/philips/PM3230/PM3230.pdf

[3] Philips IC2000 radio : https://nvhrbiblio.nl/schema/Philips_22IC2000.pdf

[4] Historical Study collection of the faculty of Electrical Engineering, Mathematics and Informatics of the University of Twente: http://studieverzameling.utwente.nl/

[5] Philips PE4804 documentation (Dutch): https://nvhrbiblio.nl/schema/Philips_PE4804.pdf

[6] Philips PE4804-05 documentation (Dutch): https://nvhrbiblio.nl/schema/Philips_PE4804-05.pdf

[7] Replacing selenium bridge rectifiers: https://yarchive.net/electr/selenium_rectifiers.html

[8] Sleepless nights from a pile of onions (Dutch): www.telegraaf.nl/nieuws/1228796/slapeloze-nachten-door-berg-uien

[9] Metal whiskers (Wikipedia): https://en.wikipedia.org/wiki/Whisker_(metallurgy)

[10] The OC171 Mystery: www.elektormagazine.com/magazine/elektor-200807/18873

[11] The AF11x problem: www.markhennessy.co.uk/articles/vintage_transistors.htm#the_af11x_problem

Fair Algorithms

By **Tessel Renzenbrink** (Elektor Ethics)

More and more attention is being paid to the responsible use of algorithms. How do we ensure that algorithms do not reproduce prejudices? And how can we guarantee that algorithms reflect our ethical values?

Cathy O'Neil, data analyst and author of the book *Weapons of Math Destruction* [1], welcomes the increasing attention paid to these issues. "This is an awkward slow frustrating conversation that we absolutely must have", O'Neil said in an interview. In mid-April she was in Amsterdam to advise the Municipality on the use of a self-learning algorithm for handling complaints.

With her book, O'Neil has made an enormous contribution to raising awareness about algorithms. Using true examples, she explains in an understandable and convincing way how mathematical models can cause damage. The weapons of math destruction (WMDs) in the title have three characteristics: they are harmful, operate on a large scale and are opaque. She writes about the latter property in her book: "verdicts from wmds land like dictates from the algorithmic gods". The results are in no way open to discussion and no one has to account for any harmful consequences.

One of the WMD's in O'Neil's book is the use of solvency for determining the car insurance premium. In the mathematical models used by American insurers to determine the price, solvency outweighs driving behaviour. In Florida, people with a good score for credit and a conviction for drink-driving paid more than $1,500 less a year than people with a poor score for credit and impeccable driving behaviour. O'Neil shows time and again that it is mainly the poor and weak groups in society who are the victims of WMDs.

Automating nepotism

However, O'Neil is not only negative about the applications of Big Data. On the contrary, they can make a positive contribution to the way in which we make decisions if we deal with them in the right way. The challenge is to ensure that mathematical models reflect our ethical values. Also, they should no longer be opaque. People must be held accountable for any negative consequences. To achieve this, algorithmic applications must be carefully tested.

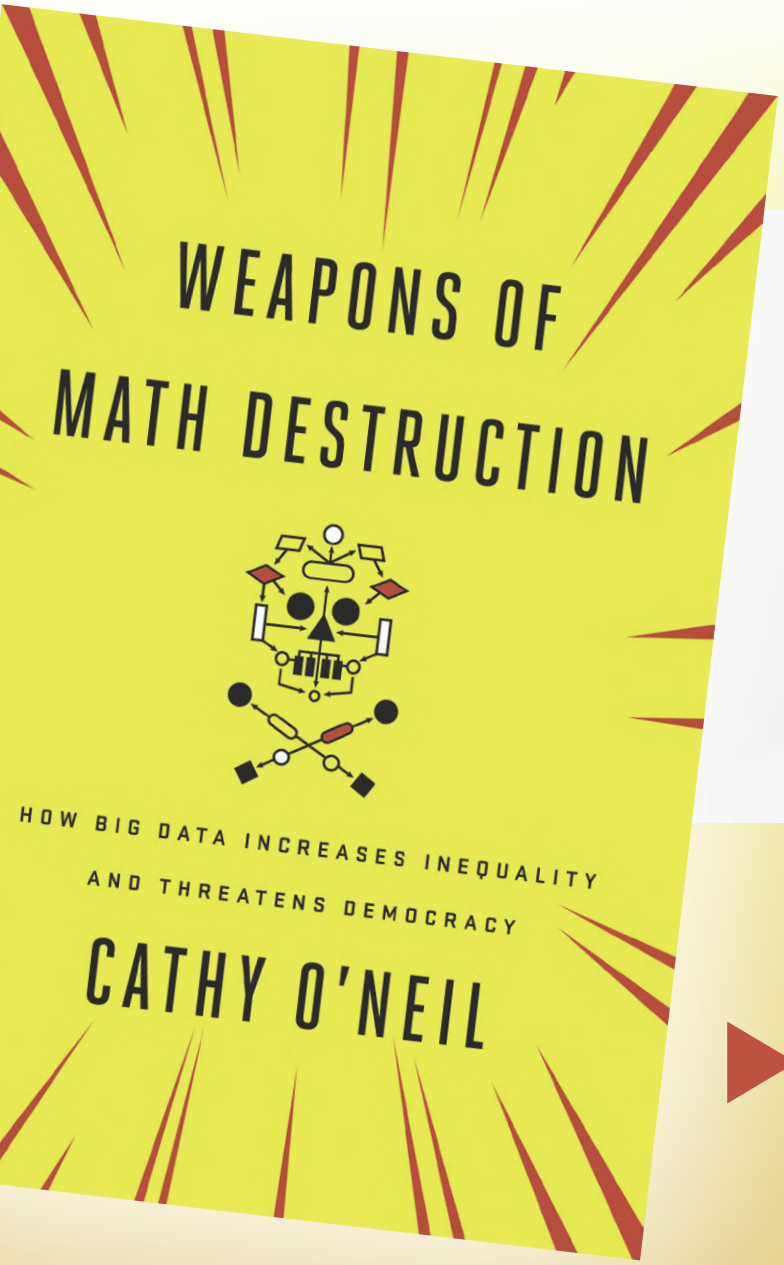
"Scrutiny is exactly what people are trying to avoid when they introduced the algorithm", O'Neil said during the interview. " The algorithm was to good to be true in a certain sense for a lot of companies and governments and organisations because it made things more efficient *and* it made accountability much less. I think of the use of algorithms as an automated decision making process. You already had decision making processes. And they are made by humans and they had all kinds of biases, all kinds of unwritten rules, all kinds of secrets and like ways of favouring their friends and stuff like that. Once it's automated they thought: 'Oh, it is like what we did but better because we also don't have to answer for it'."

Algorithms scrutinised

But the tide is turning. More and more parties are thinking about how we can make algorithms fair. One of these parties is the Municipality of Amsterdam. It is in the process of developing an algorithm to deal with complaints. Citizens' complaints about, for example, garbage in the street are automatically forwarded to the cleaning department. But the municipality wants to prevent that the automation has unintended negative consequences. One risk is that it creates inequality [2]. In richer neighbourhoods, citizens are often more vocal and are more likely to complain. If the algorithm is not adjusted, it will send sweepers to those neighbourhoods more often on the basis of that data. As a result, areas where there is less complaining are served less well.

In order to prevent such unintended effects, Amsterdam has decided to trial the algorithm. Together with the consultancy firm KPMG, the municipality is working on the development of audits for algorithms [3]. This was the reason for O'Neil's visit to Amsterdam. She advises both parties on what such an audit could look like. Her company ORCAA is specialised in performing audits on algorithms. They are checked for characteristics such as bias, transparency, accuracy and honesty.

O'Neil emphasises that developing honest algorithms is not an easy task. There is no silver bullet solution that can justify all mathematical models in one fell swoop. To ensure quality, each algorithm must be tested within the context in which it is applied. If the Amsterdam algorithm works well, it cannot simply be used in New York because the situation there is different.
"That is the biggest gap in people's





Cathy O'Neil: There is still a lot of work to be done, but we will succeed

understanding", says O'Neil. "They think they can evaluate an algorithm. They can't. They can only evaluate an algorithm in a context."

Clean up human errors

There is still a lot of work to be done, but O'Neil is positive because there is a movement of people who focus on this issue. "What this movement is going to eventually create is a way of maintaining accountability in algorithmic processes. In fact, it may even be that accountability will be much stronger than it has ever been. Once we scrutinise decision making processes, it is going to be much more transparent, much more value-laden."

O'Neil illustrates her point with an example. "Amazon's algorithm for hiring, they figured out it was sexist. So they decided not to use it. I am glad they checked, okay, that's progress. But what does it mean when you codify a human process and discover that it is sexist? It means that your human process is sexist. What I would've liked to see them do is for them to say: we have this algorithm, it is sexist, we are going to improve it. And we are going to use it. Because it will be less sexist than the human process it is replacing. But we haven't gotten to that step yet. But ideally we will. Than you are actually going to clean up, rather than just propagate, the mistakes we humans make." ⏮

180572-D-02

(Graphics: https://weaponsofmathdestruction.com)

Web Links

[1] Weapons of Math Destruction: https://weaponsofmathdestructionbook.com

[2] https://fd.nl/ondernemen/1291305/amsterdam-wil-eerlijke-computers-in-de-stad

[3] https://home.kpmg/nl/nl/home/media/press-releases/2018/12/kpmg-ontwikkelt-ai-in-control-om-gebruik-van-algoritmen-te-toets.html

welcome in your ONLINE STORE

EDITOR'S CHOICE



LabNation WiFi Bridge for SmartScope

The SmartScope is an exceptionally handy and versatile two-channel USB oscilloscope that works with all computer systems, tablets and smartphones that operate on Windows, OS X, iOS, Android and Linux. A handy expansion module is now available that enables the SmartScope to operate wirelessly via WiFi: the WiFi Bridge.

The WiFi Bridge from LabNation is a great peripheral for the SmartScope and is very interesting for anyone who would like to measure wirelessly across some distance. Very recommended to any owners of a SmartScope!

Harry Baggen
(Elektor Labs)



www.elektor.com/wifi-bridge-smartscope

Elektor Bestsellers

1. SDR Hands-on Book
   www.elektor.com/sdr-hands-on-book



2. Raspberry Pi 3 B+
   www.elektor.com/rpi-3-plus
3. Elektor SDR Shield 2.0
   www.elektor.com/170515-91
4. Bakeey UM25C USB Multimeter Logger
   www.elektor.com/bakeey-um25c
5. Miniware TS100 Soldering Iron
   www.elektor.com/miniware-ts100
6. PCBite Kit
   www.elektor.com/pcbite-kit

Embedded in Embedded

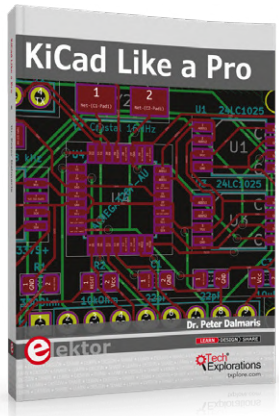


This book is a case study in embedded design including discussion of the hardware, processor initialization, low-level driver development, and application interface design for a product. Though the author describes this through a specific application of a Cortex-M3 development board, his mission is to help the reader build foundational skills critical to being an excellent product developer.

Member Price: £38.95 • €44.96 • $49.95

www.elektor.com/embedded-in-embedded

KiCad Like a Pro

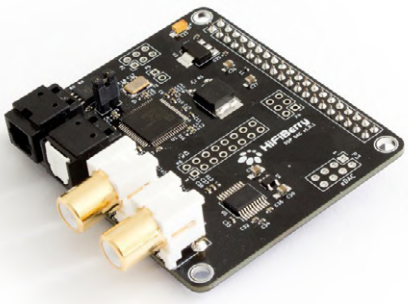


This book will teach you to use KiCad. Whether you are a hobbyist or an electronics engineer, this book will help you become productive quickly, and start designing your own boards. This book takes a practical approach to learning. It consists of four projects of incremental difficulty and recipes.

Member Price: £30.95 • €35.96 • $39.95

www.elektor.com/kicad-like-a-pro

HiFiBerry DAC+ DSP

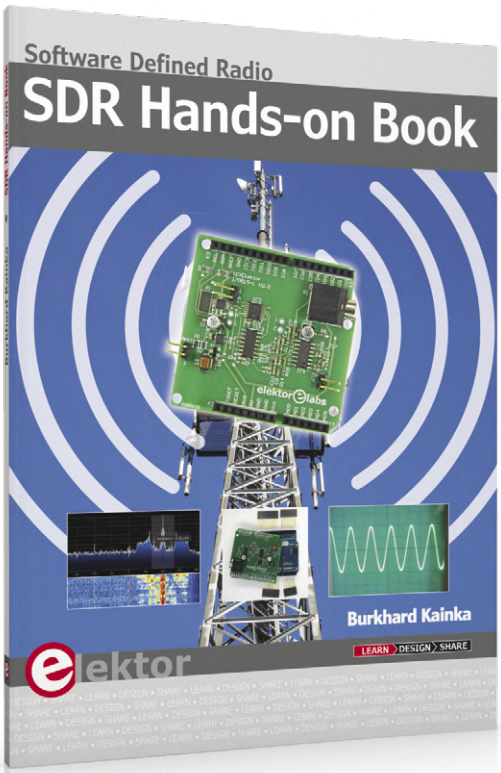


The HiFiBerry DAC+ DSP is a high-resolution digital-to-analog converter for the Raspberry Pi. It combines a Burr-Brown DAC with digital input and output and a powerful Digital Signal Processor.

Member Price: £61.95 • €71.96 • $80.95

www.elektor.com/hifiberry-dac-dsp

NEW



SDR Hands-on Book

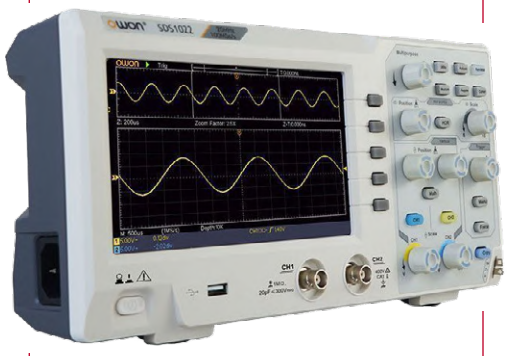
Elektor's SDR-Shield (SKU 18515) is a versatile shortwave receiver up to 30 MHz. Using an Arduino and the appropriate software, radio stations, morse signals, SSB stations, and digital signals can be received.

In this book, successful author and enthusiastic radio amateur, Burkhard Kainka describes the modern practice of software defined radio using the Elektor SDR Shield. He not only imparts a theoretical background but also explains numerous open source software tools.

Member Price: £22.95 • €26.96 • $30.95

www.elektor.com/sdr-hands-on-book

OWON SDS1022 2-ch Oscilloscope (20 MHz)



The OWON SDS1022 is a competitively priced oscilloscope with a sample rate of 1 GSa/s and a bandwidth of 20 MHz. Furthermore, with this oscilloscope you have 2 channels at your disposal and it is possible to store 10 K points per measurement. There is a USB connector on the back for exporting measurement results to the PC, and a USB port on the front for exporting to a USB memory stick.

Member Price: £147.95 • €170.96 • $191.95

www.elektor.com/owon-sds1022

The Complete ESP32 Projects Guide



The main aim of this book is to teach the Arduino IDE and MicroPython programming languages in ESP32 based projects, using the highly popular ESP32 DevKitC development board. Many simple, basic, and intermediate level projects are provided in the book using the Arduino IDE with ESP32 DevKitC. All projects have been tested and work. Block diagrams, circuit diagrams, and complete program listings of all projects are given with explanations.

Member Price: £30.95 • €35.96 • $39.95

www.elektor.com/esp32-projects-guide

PCBite Kit incl. 4 PCBite Probes and Test Wires



PCBite is the perfect tool for handling your PCB during the design process. Powerful magnets combined with a stainless-steel base plate provide flexibility, mobility and user-friendliness. The holder can easily be relocated to deal with printed circuit boards of different shapes and sizes. The probe is sturdy yet flexible and can be used for immediate measurements or totally hands-free operations with your multimeter or another preferred tool.

Member Price: £103.95 • €121.46 • $135.95

www.elektor.com/pcbite-kit

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 **$70.00 / £40.00 / €50.00 each**, which should encourage all Elektor readers to participate.

Participate!

Ultimately July 1, 2019, 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 3/2019 (May & June) is: **46F7C**.

The €50 / £40 / $70 book vouchers have been awarded to: Demierbe Guillaume (Belgium); Alexandr Papazyan (Russia); Fostiropoulos Michalis (Greece); Sune Johansson (Sweden); Ruth Hanselmann (Germany).

Congratulations everyone!

6			D	3		A		5						C	4
			C					E	3		9	0		5	6
		F			5	1	6						2		
5	1					9	B	6		C	F			3	
8						3		9	7		0	F		D	
		3			A	F	7	4		B		2			
A		D	7	2	6						E			4	
		4	E		B					2	3	6		8	0
1	6		F	4	2					0		C	8		
	2			5						3	4	1	0		D
			0		1		8	F	A	6			4		
	C		A	D		7	0		2						F
	8			A	C		F	D	6					0	2
		C						0	9	E			3		
F	D		6	0		B	3					E			
0	7						2		F		C	D			8

B	A	E	3	5	1	0	8	4	F	C	7	D	9	2	6
D	F	1	4	6	9	7	A	E	5	2	3	B	0	8	C
0	2	7	5	D	3	C	F	B	6	8	9	E	4	1	A
9	C	6	8	E	2	B	4	A	D	0	1	3	5	F	7
A	7	4	B	9	0	D	E	6	2	3	F	C	8	5	1
C	1	D	6	B	4	2	3	5	E	7	8	9	A	0	F
2	3	8	0	7	5	F	1	9	A	D	C	4	B	6	E
E	5	9	F	8	6	A	C	0	B	1	4	2	3	7	D
1	B	5	C	3	7	8	D	2	0	9	E	F	6	A	4
3	D	0	E	A	B	9	2	1	4	F	6	5	7	C	8
4	6	F	7	C	E	1	5	3	8	A	D	0	2	9	B
8	9	A	2	0	F	4	6	7	C	B	5	1	D	E	3
F	E	2	D	1	A	3	9	8	7	4	0	6	C	B	5
6	8	B	9	F	D	E	0	C	3	5	A	7	1	4	2
5	4	C	1	2	8	6	7	D	9	E	B	A	F	3	0
7	0	3	A	4	C	5	B	F	1	6	2	8	E	D	9

The competition is not open to employees of Elektor International Media, its subsidiaries, licensees and/or associated publishing houses.

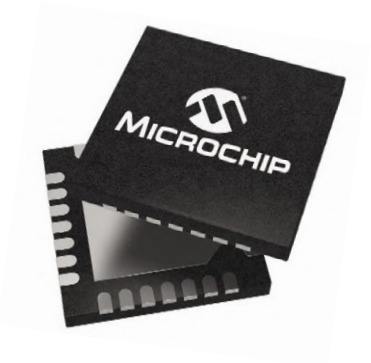


Differentiate Your Application

Customise Your Embedded Design with Flexible, Integrated Building Blocks

Microchip understands that your design needs are unique. That's why our products offer a robust set of integrated peripherals for a wide variety of applications:

- Add impactful displays and touch for intuitive user interaction
- Connect your application to the world—with and without wires
- Control your motor or power conversion
- Protect the data in your application



Customise your product with built-in peripherals and save on design costs and time.

Customise your design at www.microchip.com/FlexibleFunctions

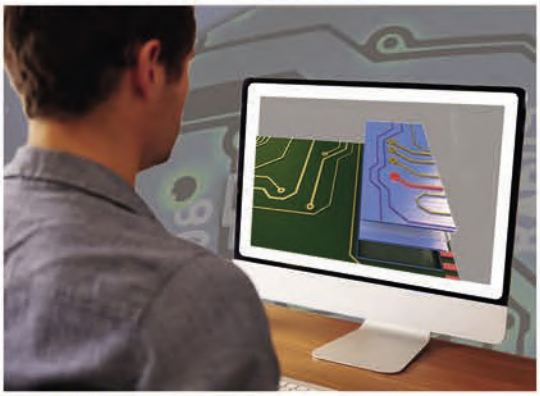
MICROCHIP

The Microchip name and logo and the Microchip logo are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries. All other trademarks are the property of their registered owners.
© 2019 Microchip Technology Inc. All rights reserved. DS30010187A. MEC2287Eng04/19

PROTEUS DESIGN SUITE

Advanced PCB features for professional board design

DESIGN ROOMS



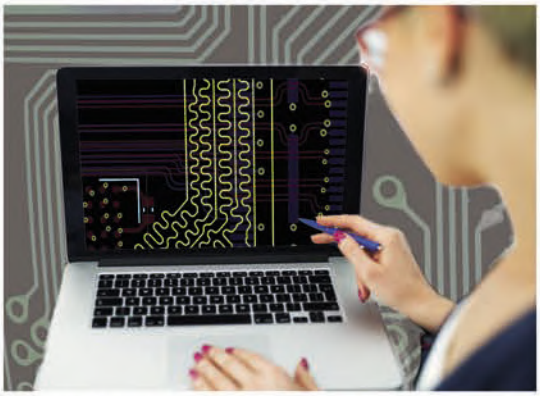
Set design rules that apply in user specified areas of the PCB.

LAYER STACKUP



Control the layer stackup and drill ranges for smarter routing.

SERPENTINE ROUTING



Easily length match tracks against each other or to a target distance.

DESIGN VARIANTS



Edit the fitted status of parts or replace with pin compatible alternatives.

The Proteus Design Suite provides advanced features at an affordable price. Try it today!

Visit: www.labcenter.com

Tel: +44 (0) 1756753440
E-Mail: info@labcenter.com
youtube.com/c/LabcenterElectronicsLtd

Labcenter Electronics www.labcenter.com