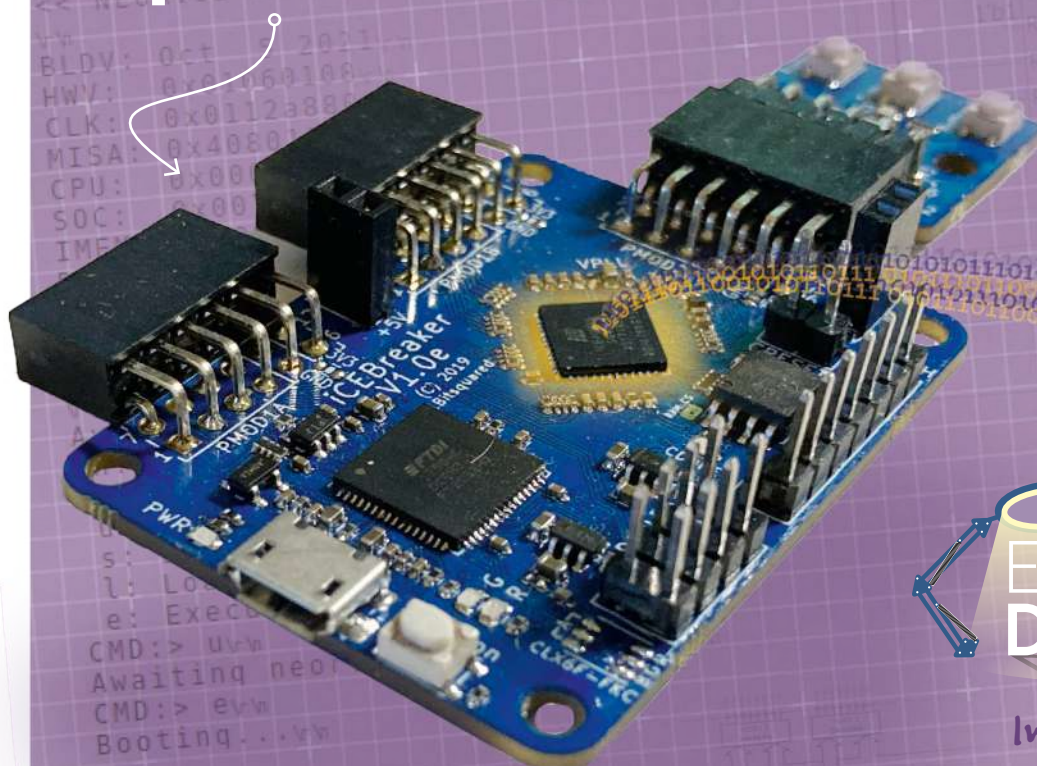


RISC-V: Build Your Own Open-Source Processor



FOCUS ON
Embedded Development

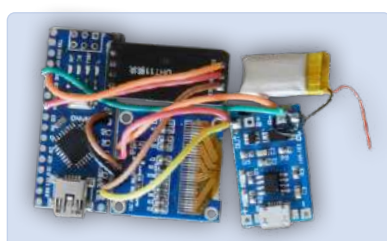
In this edition

- > How to Use Arduino's Serial Plotter
 - > Monitor and Debug Over the Air
 - > Component Identification Tips and Tricks
 - > Lithium Battery Pack Repair
 - > DIY Electronic Security and Espionage
 - > PLC Programming with the Raspberry Pi and the OpenPLC Project
 - > Magnetic Levitation the Very Easy Way
 - > Ethics: Three Questions to Build On
- and much more!

p. 52 **What's New in Embedded Development?**

Rust and Keeping IoT Deployments Updated

p. 86 **Under Your Radar**
Microcontrollers You Should Know About



Temperature- and Humidity-Measuring Device A Portable Weather Station Kit

p. 96



Buffer Board for the Raspberry Pi 400 Protect the I/Os

p. 24



DIY Touchless Light Switch It Interprets Hand Gestures!

p. 42

elektor e-zine

Your dose of electronics



Every week that you don't subscribe to Elektor's e-zine is a week with great electronics-related articles and projects that you miss!

So, why wait any longer? Subscribe today at www.elektor.com/ezine and also receive free Raspberry Pi project book!



What can you expect?

Editorial

Every Friday, you'll receive the best articles and projects of the week. We cover MCU-based projects, IoT, programming, AI, and more!

Promotional

Don't miss our shop promotions, every Tuesday and Thursday we have a special promotion for you.

Partner mailing

You want to stay informed about the ongoing activities within the industry? Then this e-mail will give you the best insights. Non-regular but always Wednesdays.

Elektor Magazine,

English edition

Edition 2/2022

Volume 48, No. 512

March & April 2022

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

www.elektor.com

www.elektormagazine.com

Elektor Magazine, English edition
is published 8 times a year by

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

Head Office:

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

Memberships:

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

Advertising & Sponsoring:

Raoul Morreau
Phone: +31 (0)6 4403 9907
E-mail: raoul.morreau@elektor.com

www.elektor.com/advertising

Advertising rates and terms available on request.

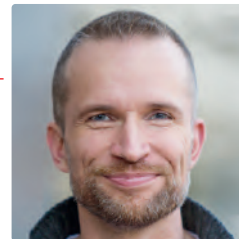
Copyright Notice

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. 2022
Printed in the Netherlands

Jens Nickel

International Editor-in-Chief, Elektor Magazine



Postponed Is Not Cancelled

For many electronics professionals, the end of winter is traditionally marked by the *embedded world* trade fair, which has taken place in previous years in late February or early March. The well-known event tends to be more informal than the huge *electronica* trade fair, which is why it's a must-attend event for everyone who deals with microcontrollers and software professionally. (And which developers don't deal with both these days?) I wish I could say that you can visit us at the Elektor booth in March this year, but the Coronavirus has thwarted our plans again. Fortunately, the *embedded world* management team recently announced that the event is "postponed is not cancelled." With new dates of June 21 to 23, 2022, the Nuremberg Exhibition Center is making a second attempt. We're knocking on wood!

In 2021, with the *embedded world* trade fair in mind, we chose the theme "Embedded Development" for our second 2022 edition of Elektor. We couldn't and didn't want to change that. So, for the May/June issue, we will focus on the Internet of Things, whose publication date will likely coincide with *embedded world* event now scheduled for June. The IoT issue will also include in-depth articles about new products and trends that will be presented in Nuremberg.

What can you expect from the issue in your hands? We present a variety of projects and background articles about small, computing chips. On page 52, my colleague Stuart Cording reports on two new trends in the embedded world, the secure programming language Rust and the management of firmware à la Toit. Starting on page 6, Elektor Lab engineer Mathias Claussen explains how to "build" and operate your own RISC-V processor. For beginners and makers, we offer a handy overview of the inexpensive boards based on the Raspberry Pi RP2040 (page 28), as well as a small workshop on the serial plotter of the Arduino IDE (page 15). We also include projects such as a buffer board for the Raspberry Pi (page 24), a non-contact light switch (page 42), and a wireless/serial interface (page 91).

Develop with us!

The Team



International Editor-in-Chief: Jens Nickel

Content Director: C. J. Abate

International Editorial Staff: Eric Bogers, Jan Buiting, Stuart Cording, Rolf Gerstendorf,
Dr Thomas Scherer, Clemens Valens

Laboratory Staff: Mathias Claussen, Ton Giesberts, Luc Lemmens, Clemens Valens

Graphic Design & Prepress: Giel Dols, Harmen Heida

Publisher: Erik Jansen



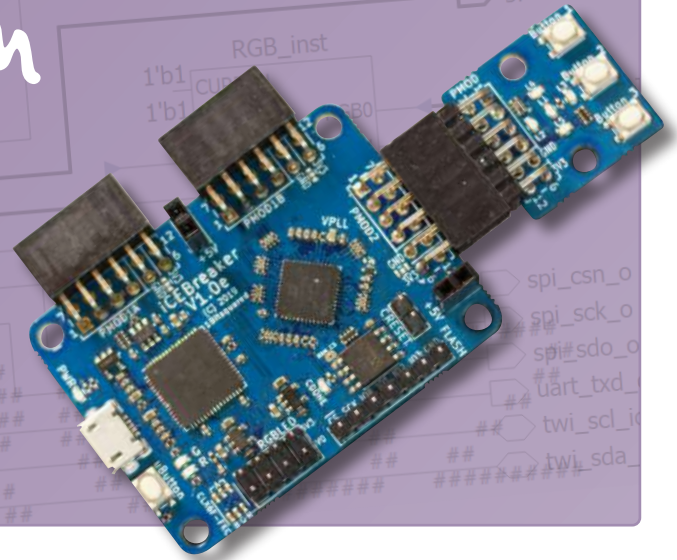
Elektor is a member of FIPP, an organization that has "grown over almost 100 years to include media owners and content creators from across the world."



Elektor is a member of VdZ (Association of German Magazine Publishers), which "represents the common interests of 500 German Consumer and B2B publishers."

Build Your Own RISC-V Controller

First Steps with the NEORV32
RISC-V Softcore for Low-Cost FPGAs



Regulars

- 3 Colophon**
- 37 Developer's Zone**
Component Identification
- 49 Starting Out in Electronics**
Matching and Transforming
- 62 Peculiar Parts**
Moving Coil Relays
- 66 HomeLab Tours**
Everything Revolves Around the Tools...
- 84 From Life's Experience**
Pack Up and Leave
- 112 Ethics**
Three Questions to Build On
- 114 Hexadoku**
The Original Elektorized Sudoku
- 68 Understanding the Neurons in Neural Networks (Part 4)**
Embedded Neurons
FOCUS
- 86 Under Your Radar**
Microcontrollers You Should Know About
- 101 Lithium Battery Pack Repair**
Save Money + More Power!
- 106 GUIs with Python**
Meme Generator

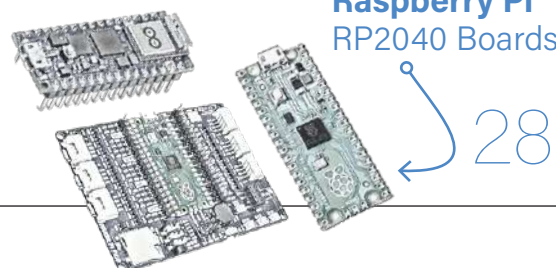
Industry

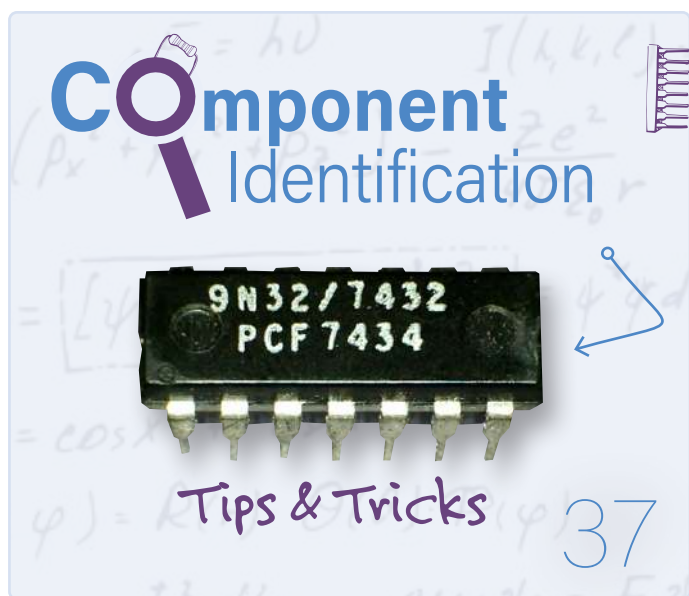
- 52 What's New in Embedded Development?**
Rust and Keeping IoT Deployments Updated
FOCUS
- 58 Infographics**
Facts and Figures About the Embedded Market
FOCUS
- 60 How the Industrial and Automotive sectors Will Benefit from 5G**

Features

- 15 How to Use Arduino's Serial Plotter**
Plotting Graphs With Arduino Is Easy
FOCUS
- 18 CLUE from Adafruit**
A Smart Solution for IoT Projects
- 28 Raspberry Pi RP2040 Boards Aplenty**
FOCUS
- 32 A Handbook on DIY Electronic Security and Espionage**
SRAM Heated or Deep-Frozen

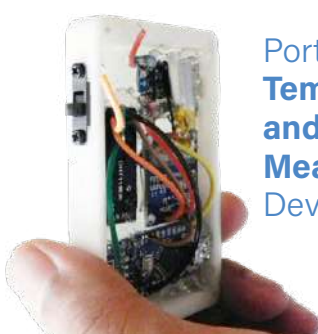
Raspberry Pi
RP2040 Boards





Projects

- FOCUS**
6 Build Your Own RISC-V Controller
 First Steps with the NEORV32 RISC-V Softcore for Low-Cost FPGAs
- FOCUS**
24 Buffer Board for the Raspberry Pi 400
 Protect the I/Os
- FOCUS**
42 DIY Touchless Light Switch
- 74 Magnetic Levitation the Very Easy Way**
 The Third and Most Compact Version
- FOCUS**
79 PLC Programming with the Raspberry Pi and the OpenPLC Project
 Visualization of PLC Programs with AdvancedHMI
- FOCUS**
91 Monitor and Debug Over the Air
 A Solution for Arduino, ESP32 and Co.
- 96 Portable Temperature- and Humidity-Measuring Device**
 Using Ready-Made Modules



Portable
Temperature-
and Humidity-
Measuring
Device

96

Next Edition

Elektor Magazine Edition 5-6/2022 (May & June 2022)

As usual, we'll have an exciting mix of projects, circuits, fundamentals and tips and tricks for electronic engineers and makers. We will focus on the Internet of Things.

From the contents:

- > First Steps with an ESP32-C3 and the IoT
- > Dual Geiger-Müller Tube Radiation Sensor for Arduino
- > CO₂ Meter with Network Connection
- > Precise Light Switch
- > IoT Cloud à la Arduino
- > Design Tools for Analog Filters
- > WinUI 3: A New Graphics Framework for Windows Apps
- > NB-IoT in Practice

And much more!

Elektor Magazine edition 5-6/2022 covering May & June 2022 will be published around May 5, 2022. Arrival of printed copies with Elektor Gold Members is subject to transport. Contents and article titles subject to change.



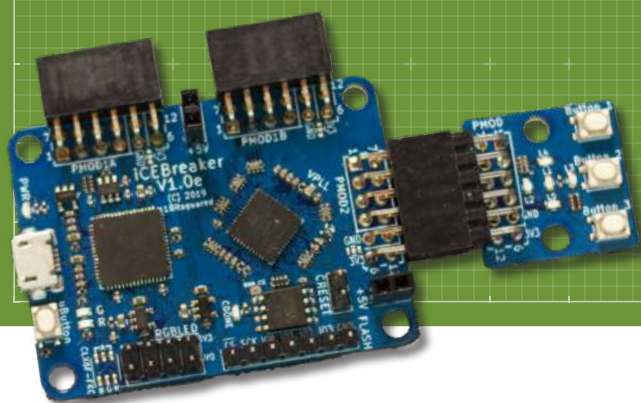
Build Your Own RISC-V Controller

First Steps with the NEORV32 RISC-V Softcore for Low-Cost FPGAs



By Mathias Claußen (Elektor)

Want to experiment with RISC-V?
You can do so without a hard-wired chip.
Get ready to work with the NEORV32
RISC-V softcore for low-cost FPGAs.



If you want to experiment with a RISC-V microcontroller, there are now a number of processors that use this open standard instruction set architecture, such as the new ESP32-C3. But you don't need to opt for a hard-wired chip. There are alternatives. The NEORV32 project offers a RISC-V softcore design which you build using an FPGA. The finished processor will be a little less powerful than a hard-wired one, but it will give you a great deal more flexibility so that different designs can be tested and any in-house developed peripherals can also be integrated into your hardware. Taking this route, you will also learn a lot, for example, about the inner workings of a CPU.

A Practical Application

Even those who have already spent some time playing with FPGAs will quickly encounter hurdles when it comes to configuring their own small processor design. The entire process can be quite challenging even for experienced engineers as our series on the SCCC project by Martin Oßmann [1] showed. Fortunately, you don't need to start from scratch. You can take advantage of some (almost turnkey) solutions already developed by dedicated experts who have made their designs freely available.

One such solution, which is also under an open-source license, will be used here. This article is in no way a comprehensive course on RISC-V or FPGA technology, but it should help shorten the learning curve by showing you how to build and get your first practical application up and running as quickly as possible.

FPGA, Synthesis, Softcore, RISC-V, and the Compiler

Whenever you need to choose a general-purpose microcontroller for a specific application, a range of different factors can influence your decision. One of the most basic considerations is the variety of built-in peripherals the controller chip offers. All these functions are fixed in the hardware of the particular version of the chip and cannot be changed. This approach allows manufacturers to produce low-cost chips with optimized performance. Things are different if you design your own controller using an FPGA.

An FPGA itself consists of a bunch of logic cells, the lookup tables (LUT), which can be flexibly interconnected via a matrix. The blocks that exist in such a LUT are shown in **Figure 1**. An example is a LUT-4 element with four input signals, a truth table, a flip-flop and a multiplexer at the output. The truth table can be used to form any basic logic gate such as an AND, OR, NOT or EXCLUSIVE OR. In conjunction with the matrix within the FPGA, these components can be used to create more complex structures such as memories, adders or multiplexers, which in turn can be combined to form an even more complex system such as a processor or a complete system-on-chip. The FPGA can be compared to a box of toy building blocks that you can plug together to build a castle, for example, and then break down to build a bridge or some other structure, using the same bricks over and over again.

In order for the FPGA to be able to perform a specific function, it must be configured appropriately. It is, however not necessary to

painstakingly create each individual LUT by hand and connect them in the matrix. This is the task of the FPGA synthesis tools. The desired functionality can be described in languages such as Verilog or VHDL. The synthesis tools can usually understand both of these hardware description languages. The synthesis tool knows the peculiarities of the FPGA and creates a bit stream from the description language, which is then used to configure the FPGA. **Figure 2** shows the rough synthesis sequence for an FPGA. Most FPGA manufacturers offer free tools that can usually work with Windows and Linux. Some FPGAs are supported by open-source solutions that can carry out this synthesis process and run preferably in a Unix-like operating systems such as Linux or macOS.

An FPGA with enough LUTs can not only map simple logic functions, but also entire processors or processor systems. These can also be described using Verilog or VHDL. Since this processor core is not permanently wired in the FPGA silicon. Its function or behavior can be adapted by modifying the hardware description. The processor unit is called a softcore. Such soft cores are available for different processor architectures. In some cases, these soft cores also contain peripherals, such as bus interfaces, etc. The open-source RISC-V processor architecture is becoming more and more popular as a softcore. The selection of hardwired RISC-V MCUs is currently still manageable, so initial experience with the architecture can be gained in this way. You can create your own RISC-V MCU to test and study it.

If you use RISC-V, there are no license fees, NDAs, or other license agreements that are usually associated with the use of other proprietary architectures. RISC-V also means that compilers to process C

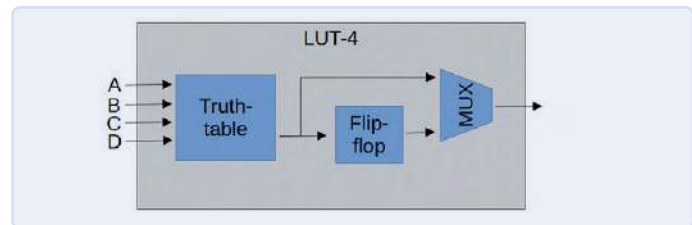


Figure 1: Signal flow in a LUT-4.

source code are already available, including in the form of the GNU C compiler (GCC). This means that the basic libraries are also in place.

NEORV32

The NEORV32 project by Stephan Nolting [4] shows that you do not need to opt for an expensive FPGA in order to build your own System on Chip (SoC). The NEORV32 is a RISC-V-compatible CPU with all the peripherals in order to run as a microcontroller-like SoC on an FPGA. The project is completely implemented in platform-neutral VHDL and is therefore not tied to individual FPGA manufacturers. The NEORV32 is not only completely open-source, it also comes with extensive documentation, a software framework and tools.

The implemented peripheral modules can be seen in **Figure 3**. SPI, I²C and UARTs are available as well as GPIOs, PWM units and a WS2812 interface. This gives beginners and advanced users a complete system including a complete development environment for the NEORV32 with all the necessary libraries for the hardware and peripherals. In addition, sample configurations are already available for some FPGA boards, so you can start without any major problems. But which of these boards run "out-of-the-box"? And how difficult is it to get the NEORV32 onto an FPGA board that is not directly supported?



Figure 2: Steps in the FPGA synthesis process.

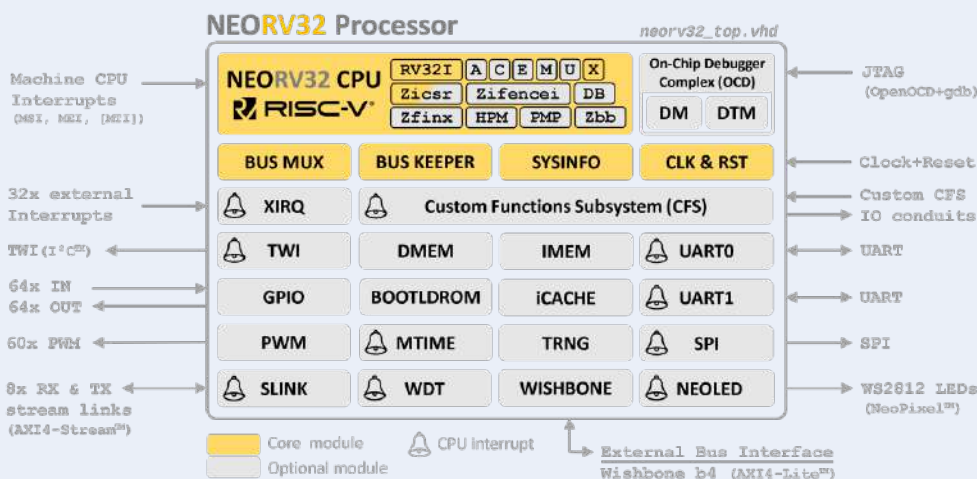


Figure 3: Block diagram showing the NEORV32 function blocks (Source: Github / Nolting, S. [20]).

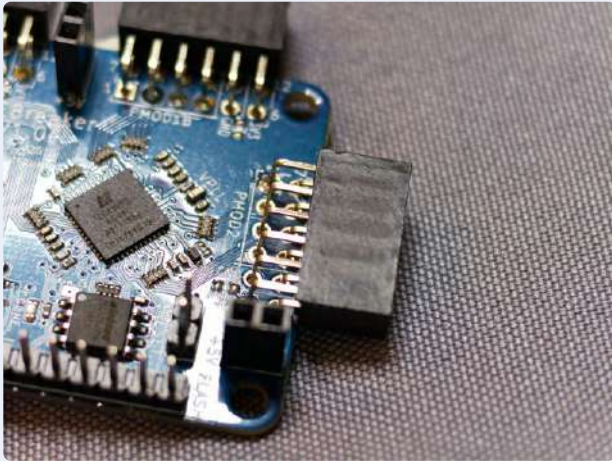


Figure 4: The iCE40UP5K in QFN outline measuring 7x7 mm.

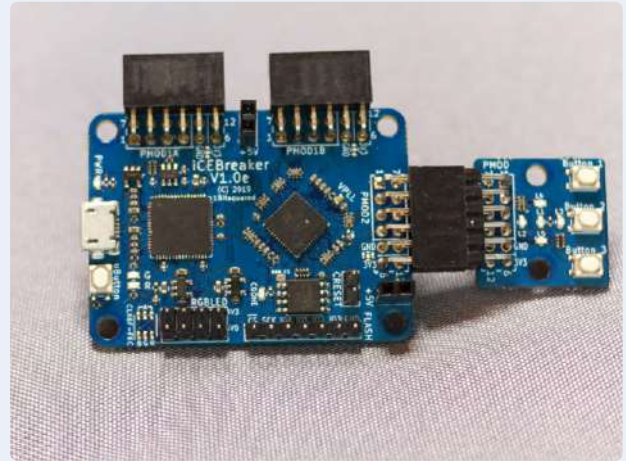


Figure 5: iCEBreaker-Board with PMOD-Header.

Pick an FPGA

In principle, any existing board with FPGA that offers enough resources can be used, as the NEORV32 does not use any manufacturer-specific extensions. One FPGA that is fitted to several inexpensive boards is the Lattice iCE40UP5K [5].

The Lattice iCE40UP5K FPGA is currently the largest version of the iCE40 Ultra-Plus variants. Altogether it has 5280 LUTs, 120 kBit (15 kByte) EBR RAM, 1024 kBit (128 kByte) SPRAM and hard-wired functional units for SPI and I²C which form a solid platform on which you can build your own projects. It's not only these features that make this FPGA interesting but also the type of chip package and its low cost. A 7 x 7 mm QFN-48 outline (**Figure 4**) is much easier to work with than a chip with a BGA outline and priced at around € 5 per chip, puts it in an interesting price bracket. Currently (October 2021) the FPGA is listed at around € 5 to € 6 per unit at all the distributors but unfortunately out of stock with a delivery time of up to 46 weeks.

For our purposes you will not need to design a PCB for the FPGA. The iCEBreaker FPGA board (**Figure 5**) and the iCEBreaker Bitsy (**Figure 6**) from 1BitSquared [6] are two open hardware dev boards on which the iCE40UP5K FPGA comes already mounted. Another option with open hardware is the UPduino V3.0 [7] from tinyVision.ai (**Figure 7**).

The NEORV32 project fully supports the UPduino V3.0 board and includes some additional sample projects. Any changes required for the iCEBreaker FPGA board can be carried out very quickly. To start off with we will use the UPduino V3.0 and then go on to show what needs to be adapted to our example project so that it runs on the iCEBreaker Board.

Using this FPGA gives you a SoC in with 64 kB space for applications, 64 kB RAM, SPI interface, I²C interface, 4 input and 4 output pins, 3 PWM pins and a UART, together with the RV32IMAC core running at 18 MHz.

The Toolchain

There are two paths you can take when it comes to selecting the toolchain for the Lattice iCE40up5k. We can install the tools from Lattice [8] or use the OSS CAD suite from YosysHQ [9] which is based

completely on open source tools. For this project, we will choose the latter, open source route. In this case it means Ubuntu 20.04 LTS will be our operating system and the OSS CAD Suite will be used to synthesize the NEORV32 for the iCE40UP5K.

In addition to the tools for the FPGA, a demo program to run on the synthesized RISC-V processor will also be compiled later: A classic implementation of the 'Hello World' message will be sent out using the UART. Here, too, open source tools are used to generate a suitable tool chain (similar to that for the Kendryte K210 [10]). A GNU C compiler (GCC) and additional libraries for the NEORV32 peripherals are available.

Mise en Place

As my colleague Clemens Valens demonstrated in his video [11], working with FPGAs is a bit like preparing a meal. The first step is to make sure you have all the ingredients and necessary utensils (tools). If you don't want to risk affecting your main operating system installation, you can also implement a virtual machine. A freshly installed version of Ubuntu 20.04 on an AMD64 system is the setup assumed here. The use of a Raspberry Pi should also be possible, since both Ubuntu 20.04 and the Raspberry Pi OS are based on Debian, but there may be small differences due to the architecture. For this project I used Ubuntu 20.04 running on an AMD64 machine only.

In order to synthesize the "hardware" for the FPGA, the current release of the OSS CAD Suite [12] needs to be downloaded into the home folder. This file is called `oss-cad-suite-linux-x64-xxxxxxx.tgz`. In a terminal, this file is now unpacked using `tar -xvzf oss-cad-suite-linux-x64-xxxxxxx.tgz` and then moved to `/opt` with `sudo mv ~/oss-cad-suite /opt/`. In order that the folder can be accessed later, the rights are set by using `chmod 777 /opt/oss-cad-suite -R`. The `libgnat-9` library is also required which is installed with `sudo apt install libgnat-9`. The FPGA tools are now in place.

Compiler

Next we need the files associated with the NEORV32 [4] from the GitHub repository. To do this, we will install `git` by entering `sudo apt install git`. The NEORV32 repository is then cloned with `git clone https://github.com/stnolting/neorv32.git ~/neorv32`.

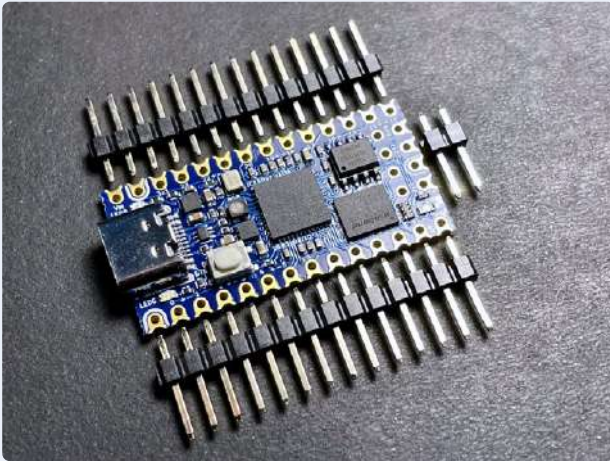


Figure 6: iCEBreaker Bitsy (Source: https://cdn.shopify.com/s/files/1/1069/4424/products/IMG_3859_large.jpg / 1BitSquare).

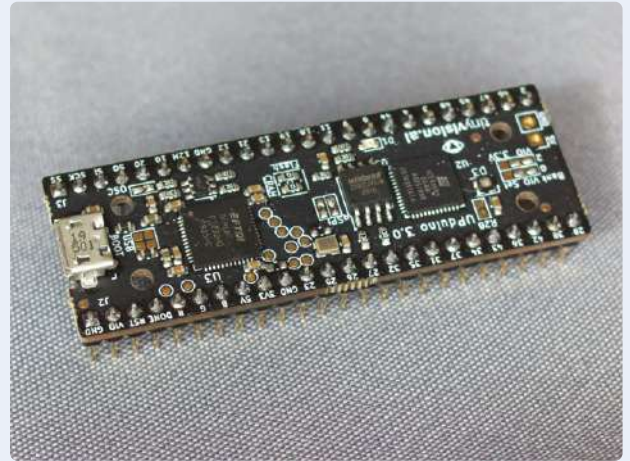


Figure 7: The UPduino V3.0 with pin header strips fitted.

So that we will be able to communicate with the NEORV32 later, a terminal program will be required. For this, I normally use HTerm by Tobias Hammer, which has proven to be a useful tool. This program can be downloaded from his homepage [13] with `wget https://www.der-hammer.info/terminal/hterm-linux-64.tgz`. Now with `mkdir ~/hterm && tar -xvf hterm-linux-64.tgz -C ~/hterm` the contents of the tgz file will be extracted to the `~/hterm` folder. So that the serial interfaces can be accessed later by a user, they must be added as a member to the `dialout` group. To do this use the terminal and enter `sudo adduser $(whoami) dialout`.

The C compiler itself now needs to undergo a process of compilation. In the examples for the iCE40up5k, the NEORV32 is configured as RV32IMAC, so that commands for integer multiplication and division are available (see also the **RISC-V Naming Convention** box). The compiler must be compiled in accordance with this particular architecture and the command extensions; you can read in the NEORV32 [14] documentation why this adaptation is important.

Using a terminal, first go to the home folder using `cd ~` and enter `git clone https://github.com/riscv/riscv-gnu-toolchain --recursive` to clone the RISC-V Toolchain. We will also need some additional packages which can be installed using the following:

```
sudo apt-get install autoconf automake autotools-dev
curl python3 libmpc-dev libmpfr-dev libgmp-dev gawk
build-essential bison flex texinfo gperf libtool
patchutils bc zlibg-dev libexpat-dev
```

Finally, the compiler and libraries can be compiled.

With RISC-V CPU models a hierarchy exists in terms of the supported commands. This means that code compiled for an RV32I processor model will also be able to run on an RV32IMAC model, but not the other way around. Therefore, the recommendation is to first compile the toolchain so that it is compatible to the lowest common denominator model. Using the terminal, we can switch to the directory containing the cloned toolchain with `cd ~/riscv-gnu-toolchain` and then use `./configure --prefix=/opt/riscv --with-arch=rv32i --with-abi=ilp32` to preconfigure the toolchain for compilation.

Now we can start the compilation process using `sudo make`. The toolchain can then be found in `/opt/riscv`. So that everyone can access the compiler, the rights to `of/opt/riscv` need to be modified. Using a terminal you can grant everyone access with the command `chmod 777 /opt/riscv -R`.

For the OSS CAD Suite and the RISC-V GCC Toolchain, it is necessary to make a modification of the path variable in the `/etc/environment` file. Using `sudo nano /etc/environment` we can open the file and then after `PATH=` enter the string `/opt/oss-cad-suite/bin:/opt/riscv/bin:` and then save the file.

Most FPGA boards use an FT232H chip from FTDI to handle the programming interface. In order for a user to gain access to it without root rights, an appropriate `udev` rule must be created for the FTDI chip. Using the terminal enter `sudo nano /etc/udev/rules.d/53-lattice-ftdi.rules` to open a new file for writing to. In this file it is necessary to enter:

```
ACTION=="add", ATTR{idVendor}=="0403",
    ATTR{idProduct}=="6010", MODE=="666"
ACTION=="add", ATTR{idVendor}=="0403",
    ATTR{idProduct}=="6014", MODE=="666"
```

Then, save the file. The preparations are now complete and the synthesis and subsequent upload of our first test program can begin. A reboot should first be carried out so that all the settings take effect. Another recommendation is to install an editor with syntax highlighting for use with VHDL and Verilog.

NEORV32 for the FPGA

In its de-energized state, the FPGA is not configured. On power up, the iCE40UP5K reads the connection description from an external SPI flash. This description of the internal connections must first be stored into the SPI flash on the UPduino V3.0 or the iCEBreaker.

In this article, we create a sample project for the UPduino V3.0 board, which contains the processor and peripherals. This creates a system that should be able to run directly on the FPGA.

```

MEMORY
{
  /* section base addresses and sizes have to be a multiple of 4 bytes */
  /* ram section: first value of LENGTH => data memory used by bootloader (fixed!); second value of LENGTH => *physical* size of data memory */
  /* adapt the right-most value to match the *total physical data memory size* of your setup */

  ram (rwx) : ORIGIN = 0x80000000, LENGTH = DEFINED(make_bootloader) ? 512 : 8*1024

  /* rom and iodev sections should NOT be modified by the user at all! */
  /* rom section: first value of ORIGIN/LENGTH => bootloader ROM; second value of ORIGIN/LENGTH => maximum *logical* size of instruction memory */

  rom (rx) : ORIGIN = DEFINED(make_bootloader) ? 0xFFFF0000 : 0x00000000, LENGTH = DEFINED(make_bootloader) ? 32K : 2048M
  iodev (rw) : ORIGIN = 0xFFFFFE00, LENGTH = 512
}
/* ***** */

```

Figure 8: Changes necessary to configure the RAM size.

```

user@user-VirtualBox:~/neorv32/sw/example/hello_world$ make exe
Memory utilization:
text  data  bss   dec   hex filename
5576   0     116  5692  163c main.elf
Executable (neorv32_exe.bin) size in bytes:
5588
user@user-VirtualBox:~/neorv32/sw/example/hello_world$

```

Figure 9: NEORV32_exe.bin is created.

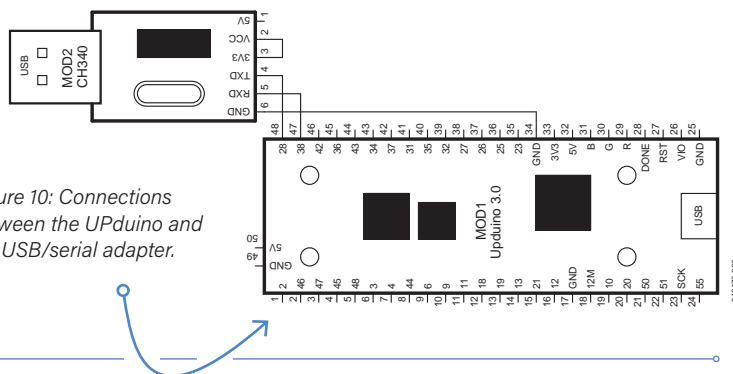


Figure 10: Connections between the UPduino and the USB/serial adapter.

In a terminal you now have to change to the example directory for the open-source tools with `cd ~/neorv32/setups/osflow/`. In order to start the synthesis and thus the creation of the bit stream for the FPGA, we just need to enter `make BOARD=UPduino UP5KDemo` as a command, after that it can take a little while until the process of synthesis is complete. Now in the `~/neorv32/setups/osflow/` folder a file named `neorv32_UPduino_v3_UP5KDemo.bit` has been created. The bit stream contained in this file describes how the basic logic blocks must be interconnected in the FPGA. This bit stream now needs to be written into the SPI flash on the FPGA board.

Now we can hook up the UPduino board to the PC via a USB cable. In the terminal we enter `icprog ~/neorv32/setups/osflow/neorv32_UPduino_v3_UP5KDemo.bit`. This starts the programming of the external SPI flash, and the configuration is then loaded into the FPGA. This means that we now have a RISC-V system configured in the UPduino V3.0, which we can now supply with software.

As mentioned at the beginning, 64 KB of memory is available for applications and 64 KB is available as RAM. For the peripherals we have an SPI interface, I²C, a UART, four inputs and four outputs, as well as three PWM outputs. The CPU synthesized here is an RV32IMAC model that runs at 18 MHz. The SoC in the FPGA also has a small bootloader that can be accessed via the UART.

Hello World

The FPGA is now equipped with the NEORV32 and the first *Hello World* demo can be compiled and uploaded to the RISC-V. The NEORV32 is configurable so the current size of the RAM must be defined appropriately in the linker script. To do this, use the terminal to enter `nano ~/neorv32/sw/common/neorv32.ld` and in line 62:

```

ram (rwx) : ORIGIN = 0x80000000, LENGTH =
            DEFINED(make_bootloader) ? 512 : 8*1024

```

In exchange for:

```

ram (rwx) : ORIGIN = 0x80000000, LENGTH =
            DEFINED(make_bootloader) ? 512 : 64*1024 (Figure 8)

```

The RISC-V compiler is now ready for use. In an open terminal you can go to the folder of the *Hello World* program by entering:

```
cd ~/neorv32/sw/example/hello_world
```

To generate an executable file that can be loaded into the NEORV32, you just need to enter `make exe` to generate the `neorv32_exe.bin` file (Figure 9). In order for the NEORV32 to be able to be executed, it should now be uploaded to the board. The integrated bootloader is used for this and receives data via the UART (19200 baud, 8 data bits, 1 stop bit, no parity, no flow control). If a UPduino V3.0 board is used, an external USB-to-serial converter will be required, such as the CH340-based one from the Elektor Store. (Refer to the **Related Products** text box.) This must be connected as shown in Figure 10.

HTerm is used for the upload itself. This can be started from a terminal with `~/hterm/hterm` a window like in Figure 11 should appear. The USB-to-serial converter must now be selected as the port, which usually reports itself as `/dev/ttyUSB0`; depending on the hardware configuration and the selected adapter, however, this may be different.

After connecting to the USB serial adapter, the UPduino can be supplied with voltage and the bootloader message should be visible (Figure 12). If a character is not sent to the bootloader in time, it tries to autoboot from the SPI flash, which currently does not contain any software. Any character sent within 8 seconds of starting the bootloader will switch it into command mode. A `u` must then be sent to activate the upload in the bootloader and the *Select File* button must be clicked in HTerm. As can be seen in Figure 13, select the file `neorv32_exe.bin` in the `~/neorv32/sw/example/hello_world` folder and then upload it. When everything is finished, the bootloader reports an `OK` as shown in Figure 14 and the program can be executed by entering `e`.

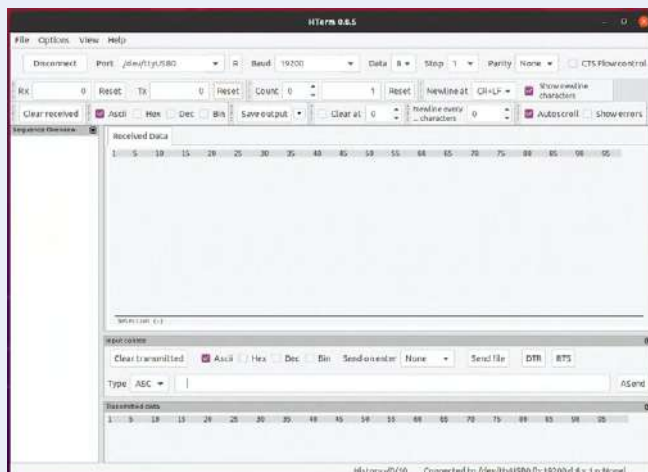


Figure 11: An open *HTerm* window.

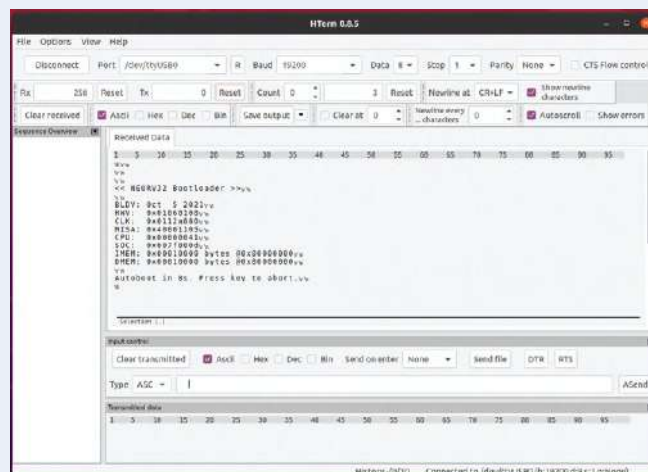


Figure 12: The NEORV32 bootloader.

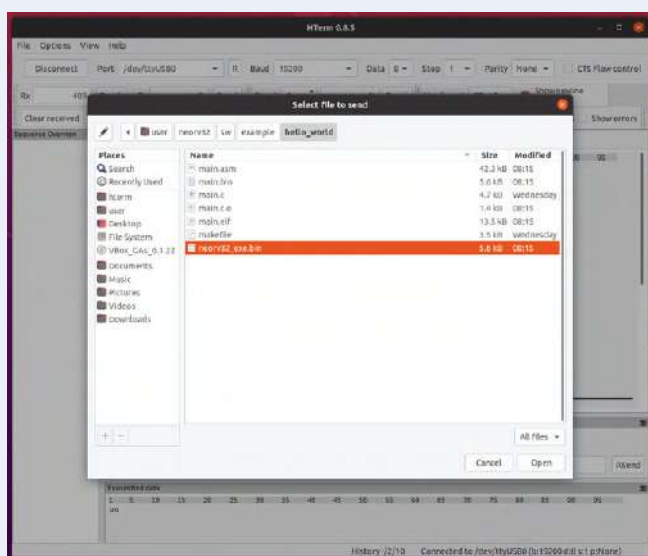


Figure 13: Dialog to upload neorv32_exe.bin.

```
Available CMDs:\n
h: Help\n
r: Restart\n
u: Upload\n
s: Store to flash\n
l: Load from flash\n
e: Execute\n
CMD:> U\n
Awaiting neorv32 exe.bin... OK\n
CMD:>
```

Figure 14: A successful upload.

[illegible]

Figure 15: ‘Hello World’ from the NEORV32.

RISC-V Naming Convention

RISC-V is a generic term for the different architecture variants. Our Elektor colleague Stuart Cording wrote a nice article ("What is RISC-V," Elektor 7-8/2021, [2]) in which the details are explained in more detail. RISC-V describes an ISA in which the processors are categorized as 32, 64 or 128 bit versions. A 32-bit processor has a designation that starts with RV32 for 32 bits, so that an RV64 indicates a 64-bit processor. In addition, after RV32 or RV64, there are a number of letters that indicate which commands and extensions the processor can handle. These letters range from A to Z; their meaning can be looked up in the current RISC-V specification [3]. The performance of the processors differ according to the supported commands and extensions.

The result can be seen in **Figure 15**. The program has not been stored in the SPI flash, but into the RAM-based "ROM" of the NEORV32. This extends the service life of the SPI flash by cutting down on the number of write cycles. The disadvantage of this configuration is that, each time the NEORV32 is restarted, the program needs to be uploaded again. If the program is to be loaded automatically, the bootloader must upload it to the SPI flash. In addition to the basic "Hello World" demo, there are other examples to discover, including a complete FreeRTOS, which has already featured in Elektor [15]. It is worth taking a look at the `~/neorv32/sw/example` folder, where several other examples can be found in their own folders.

A New iCE40UP5K FPGA Environment

Using the iCEBreaker Board will demonstrate how the NEORV32 can also be adapted to other iCE40up5k boards. The features of the SoC itself are not changed here, but the pinout assignment on the FPGA is adapted so that an iCEBreaker board can be used and a suitable bit stream will be generated.

To do this, the Makefile in `~/neorv32/setup/osflow` needs to be edited. The file is opened using your text editor of choice and the new board is added as the target. For the iCEBreaker board we write the following in line 72:

```
iCEBreaker:
$(MAKE) \
BITSTREAM=neorv32_${(BOARD)}_${(DESIGN)}.bit \
NEORV32_MEM_SRC="devices/ice40/neorv32_imem.ice40up_
sram.vhd devices/ice40/neorv32_dmem.ice40up_sram.
vhd" \
run
```

This ensures that the board is referenced in the primary Makefile. Also in `~/neorv32/setup/osflow/boards` an *iCEBreaker.mk* file with the following contents is required:

```
.PHONY: all

all: bit
echo "! Built $(IMPL) for $(BOARD)"
```

The Makefiles are thereby prepared but two VHDL-files are still missing.

In `~/neorv32/setup/osflow/board_tops/` the files *neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd* and *neorv32_iCEBreaker_BoardTop_MinimalBoot.vhd* must be generated. Since their contents are almost identical to that of the *neorv32_UPduino_BoardTop_UP5KDemo.vhd* and *neorv32_UPduino_BoardTop_MinimalBoot.vhd* files, they can just be copied and renamed. This should be done using a terminal and entering:

```
cp ~/neorv32/setups/osflow/board_tops/neorv32_UPduino_
BoardTop_MinimalBoot.vhd
~/neorv32/setups/osflow/board_tops/neorv32_iCEBreaker_
BoardTop_MinimalBoot.vhd
```

And:

```
cp ~/neorv32/setups/osflow/board_tops/neorv32_UPduino_
BoardTop_UP5KDemo.vhd
~/neorv32/setups/osflow/board_tops/neorv32_iCEBreaker_
BoardTop_UP5KDemo.vhd
```

A few changes need to be made to the two files *neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd* and *neorv32_iCEBreaker_BoardTop_MinimalBoot.vhd*. In file *neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd* must line 42 in `entity neorv32_iCEBreaker_BoardTop_UP5KDemo is` and line 68 in `architecture neorv32_iCEBreaker_BoardTop_UP5KDemo_rtl of neorv32_iCEBreaker_BoardTop_UP5KDemo is` be changed. In the file *neorv32_iCEBreaker_BoardTop_MinimalBoot.vhd* line 42 in `entity neorv32_iCEBreaker_BoardTop_MinimalBoot is` and line 54 in `architecture neorv32_iCEBreaker_BoardTop_MinimalBoot_rtl of neorv32_iCEBreaker_BoardTop_MinimalBoot is` are to be changed. The last step is for the *iCEBreaker.pcf* Constraints-File to be placed in `~/neorv32/setups/osflow/constraints/`. The contents of the file can be seen in **Listing 1**. *iCEBreaker.pcf* defines which function should be routed to which pin of the FPGA. This also directly integrates the USB/serial converter, which is on the iCEBreaker board, and routes the buttons and LEDs on the IO pins of the NEORV32. One thing that is missing is the reset button. Even though it is already defined in the constraints file, its function is not referenced here.

With all the necessary changes made, we can generate a bit stream just like we did with the UPduino board. For this we need a terminal and enter `cd ~/neorv32/setups/osflow` to get to the *osflow* folder. Using `make BOARD=iCEBeaker UP5KDemo` we can start the make process. To upload the bitstreams to the iCEBreaker (just like we did in the UPduino) we can use `iceprog ~/neorv32/setups/osflow/neorv32_iCEBreaker_UP5KDemo.bit`.

Uploading the software for the NEORV32 is also taken care of by the integrated bootloader. Using this board we will not need an external USB/serial converter; instead, we use the second channel of the converter integrated on the iCEBreaker board.

A Reset Button for the NEORV32

To restart the NEORV32, the power supply needs to be briefly disconnected and then reconnected. This gets to be a bit annoying eventually, and since the iCEBreaker has enough buttons, one of them can be



Listing 1: iCEBreaker.pcf

```
#UART (uart0)
ldc_set_location -site {9} [get_ports uart_txd_o]
ldc_set_location -site {6} [get_ports uart_rxd_i]

#SPI - on-board flash
ldc_set_location -site {14} [get_ports flash_sdo_o]
ldc_set_location -site {15} [get_ports flash_sck_o]
ldc_set_location -site {16} [get_ports flash_csn_o]
ldc_set_location -site {17} [get_ports flash_sdi_i]

#SPI - user port
ldc_set_location -site {43} [get_ports spi_sdo_o]
ldc_set_location -site {38} [get_ports spi_sck_o]
ldc_set_location -site {34} [get_ports spi_csn_o]
ldc_set_location -site {31} [get_ports spi_sdi_i]

#TWI
ldc_set_location -site {2} [get_ports twi_sda_io]
ldc_set_location -site {4} [get_ports twi_scl_io]

#GPIO - input
ldc_set_location -site {18} [get_ports {gpio_i[0]}]
ldc_set_location -site {19} [get_ports {gpio_i[1]}]
ldc_set_location -site {20} [get_ports {gpio_i[2]}]
ldc_set_location -site {28} [get_ports {gpio_i[3]}]

#GPIO - output
ldc_set_location -site {25} [get_ports {gpio_o[0]}]
ldc_set_location -site {26} [get_ports {gpio_o[1]}]
ldc_set_location -site {27} [get_ports {gpio_o[2]}]
ldc_set_location -site {23} [get_ports {gpio_o[3]}]

#RGB power LED
ldc_set_location -site {39} [get_ports {pwm_o[0]}]
ldc_set_location -site {40} [get_ports {pwm_o[1]}]
ldc_set_location -site {41} [get_ports {pwm_o[2]}]

#Reset
ldc_set_location -site {10} [get_ports {user_reset_btn}]
```

used as a reset. We can use the uButton near the micro-USB socket on the board, which is connected to pin 10 of the FPGA.

The NEORV32 has an internal reset input which is connected to the “lock” output from the system PLL. When the PLL goes out of lock, it issues a reset to the NEORV32. The cleanest solution would be to gate the user reset signal from the button together with the lock signal so that either would issue a reset to the NEORV32. The quick and dirty solution is to just use the reset input of the PLL. **Figure 16** shows the connection of the uButton with this input. Now when the PLL is reset, the NEORV32 is also reset, since the *lock_o* signal goes low when PLL is reset.

In order to incorporate this change into the UP5K-demo project, it will be necessary to insert one line and then make a change to another

line in the *neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd* file residing in the *~/neorv32/setups/board_tops* folder. After line 44, insert the new line *user_reset_btn : in std_ulogic;*. Line 130 now needs to be changed to *RESETB => user_reset_btn,*. Be careful, a syntax error will occur if you forget the comma at the end. The uButton is now configured to function as a reset. Finally, to make the changes, a new bit stream must be generated in the UP5K-Demo and loaded into the iCEBreaker FPGA.

Outlook

No doubt we could dedicate a whole book on the subject of RISC-V, FPGA, and also the NEORV32. The iCE40UP5K is an inexpensive choice for beginners, if only the chip were more widely available. The two boards used in the article are not the only ones that use this FPGA.

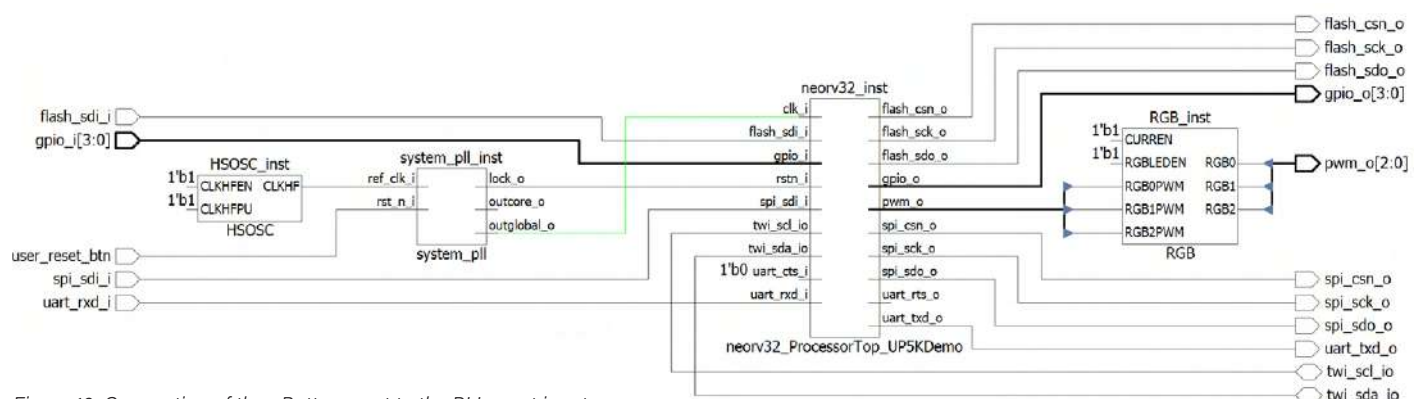


Figure 16: Connection of the uButton reset to the PLL reset input.

There is at least a handful of other platforms. From the Raspberry Pi add-on to the complete handheld console, many options are available. The various tools and projects applicable to this FPGA are also worth investigating. Examples are LiteX [16], with which your own SoC can be put together like in a construction kit and is not necessarily limited to RISC-V, the RISCboy [17] by Luke Wren, who provides a Gameboy-like system in the FPGA, and the OK-ice40-PRO [18] gamepad console.

Should the procurement situation improve, there will certainly be additional projects and ideas for the iCE40UP5K. A small games console built using an iCE40UP5K together with a Raspberry Pi RP2040 already seems to be in the pipeline [19]. ◀

210175-01

Contributors

Text and images: **Mathias Claußen**

Editor: **Jens Nickel**

Translation: **Martin Cooke**

Layout: **Harmen Heida**

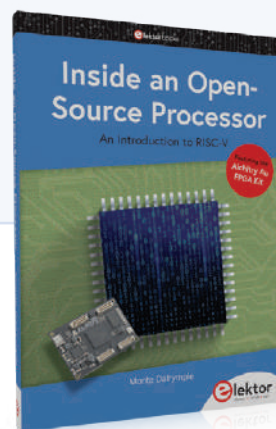
Questions or Comments?

Do you have any technical questions or comments relating to this article? Contact the author at mathias.claussen@elektor.com or contact the Elektor team at editor@elektor.com.



RELATED PRODUCTS

- **Alchitry Cu FPGA Development Board (Lattice iCE40 HX) (SKU 19640)**
www.elektor.com/19640
- **CH340 USB to TTL Converter UART Module CH340G (3.3 V/5.5 V) (SKU 19151)**
www.elektor.com/19151
- **M. Dalrymple, *Inside an Open-Source Processor* (Elektor 2021) (SKU 19826)**
www.elektor.com/19826



WEB LINKS

- [1] M. Ossmann, "The SCCC project (1)," Elektor 3-4/2019: <https://www.elektormagazine.com/magazine/elektor-88/42444>
- [2] S. Cording, "What is RISC-V," Elektor 7-8/2021: <http://www.elektormagazine.com/magazine/elektor-179/59732>
- [3] RISC-V Specification: <http://riscv.org/technical/specifications/>
- [4] NEORV32 GitHub Repository: <http://github.com/stnolting/neorv32/>
- [5] Lattice iCE40UltraPlus Product page: <http://www.latticesemi.com/en/Products/FPGAandCPLD/iCE40UltraPlus>
- [6] iCEBreaker GitHub Repository: <http://github.com/icebreaker-fpga/icebreaker>
- [7] UPduino GitHub Repository: <http://github.com/tinyvision-ai-inc/UPduino-v3.0>
- [8] Lattice Radiant: <http://www.latticesemi.com/LatticeRadiant>
- [9] YosysHQ oss-cad-suite-build : <http://github.com/YosysHQ/oss-cad-suite-build>
- [10] Setup a toolchain for the Kendryte K210, Elektor Labs:
<http://www.elektormagazine.com/labs/setup-a-toolchain-for-the-kendryte-k210-1>
- [11] Linux-flavored Snickerdoodles with Zynq, ElektorTV: <http://www.youtube.com/watch?v=EE4yYZ-EEoQ>
- [12] YosysHQ oss-cad-suite-build Releases: <http://github.com/YosysHQ/oss-cad-suite-build/releases>
- [13] HTerm: <http://www.der-hammer.info/>
- [14] NEORV32 - Build the Toolchain from scratch: http://stnolting.github.io/neorv32/ug/#_building_the_toolchain_from_scratch
- [15] W. Gay, "Practical ESP32 Multitasking," Elektor 1-2/2020: <http://www.elektormagazine.com/magazine/elektor-139/56997>
- [16] LiteX: <http://github.com/enjoy-digital/litex>
- [17] RISCBoy : <http://github.com/Wren6991/RISCBoy>
- [18] OK-iCE40Pro Handheld: <http://github.com/WiFiBoy/OK-iCE40Pro>
- [19] PicoStation3D: <http://github.com/Wren6991/PicoStation3D>
- [20] Nolting S., The NEORV32 RISC-V-Processor, GitHub repository 2020:
http://raw.githubusercontent.com/stnolting/neorv32/master/docs/figures/neorv32_processor.png

How to Use Arduino's Serial Plotter

Plotting Graphs With Arduino Is Easy

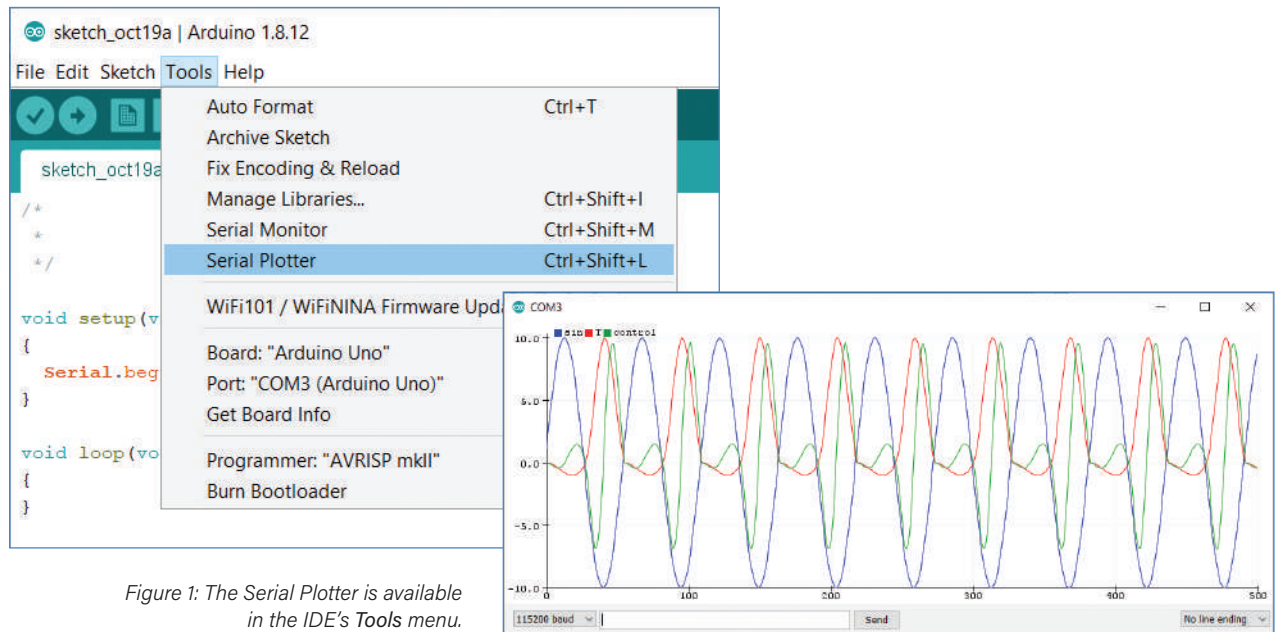


Figure 1: The Serial Plotter is available in the IDE's Tools menu.

By **Clemens Valens** (Elektor)

One of the strong points of Arduino is its easy-to-use serial port, which makes it simple to export data, to send commands to it, or to help with debugging. A Serial Monitor and a Serial Plotter are available to visualise this data. The first does not need much explaining, but how do you use the Serial Plotter?

Most Arduino users will know about the Serial Monitor built into the IDE (available from the *Tools* menu, or with keyboard shortcut *Ctrl+Shift+M*). This is a simple serial terminal that displays any data received on the serial port. It also lets you send strings (e.g., commands or data) from the computer to the Arduino board. The Serial Monitor can be handy for debugging by letting the sketch send text messages about what it is doing or what it is supposed to be doing.

The Serial Plotter

The Serial Monitor can display numerical values too, of course. However, trying to follow the evolution of a parameter when it is printed as a list of values is not always easy, which is why the Arduino IDE also

has a Serial Plotter (on the *Tools* menu, or with keyboard shortcut *Ctrl+Shift+L*, **Figure 1**).

The Serial Plotter creates graphs of the numerical data that it receives and displays them. It is not super powerful and does not have many options, but it is easy to use. Unfortunately, there is no documentation for it. You must refer to the Java source code [1] to find out what it is capable of. Doing so, you will discover the following features:

- Unlimited number of graphs;
- Automatic scaling of the vertical or Y-axis;
- Five hundred points on the horizontal or X-axis;
- Show the most recent 500 data points.

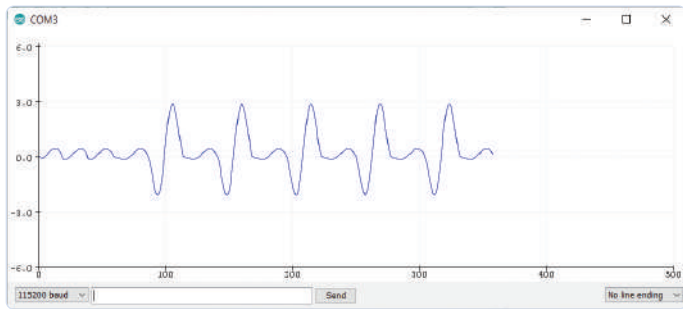


Figure 2: The horizontal axis has room for up to 500 points. If you send more, it will start scrolling.

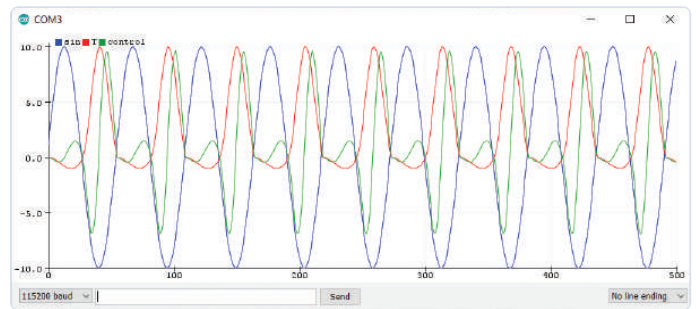


Figure 3: The number of graphs is unlimited (in theory). Use labels to keep track of which graph represents what data.

Plotting a graph

After opening the Serial Plotter, select the same baud rate as the speed specified in `Serial.begin` in the sketch. The box next to the speed selection box is intended for sending data back to the sketch, exactly as in the Serial Monitor. This allows you to control the sketch in some way.

You don't need to learn special commands to use the Serial Plotter, it makes use of the same serial port commands as the Serial Monitor. The Serial Plotter is just another way to visualize data sent over the serial port.

The only important thing is the way you format the data. To plot a graph of a single parameter (**Figure 2**), a sensor value for instance, it is enough to separate the values by a newline character. In other words, to send the values, use

```
Serial.println(sensor_value);
```

The scales are out of control

The Serial Plotter uses these values for the vertical axis, the Y-axis; it increments the horizontal or X-axis itself. In other words, the first value sent will be considered the 'y' value for $x = 0$, the second value is the 'y' value for $x = 1$, and so on, and so forth. When x reaches 500, the graph starts scrolling horizontally; new points are added at the end, the old points fall off from the beginning.

When the y-value is out of bounds, the vertical axis scales up automatically to keep everything within vertical bounds. The same is true the other way around — i.e., when suddenly all values fit in a smaller bound (because a high value fell off at the beginning when the graph is scrolling) — the scale goes down.

Multi-graph plotting

To draw graphs for two parameters or more, just send the values for the different graphs as a list of values separated by either a comma (`,`), a whitespace (` `) or a tab character (`\t`) and terminate the list with a newline character, for instance:

```
Serial.print(temperature);
Serial.print(',');
Serial.print(humidity);
Serial.print(',');
Serial.print(pressure);
Serial.println();
```

When using the comma as a separator, the format is known as 'CSV', which stands for comma-separated values. Most spreadsheets can import such files (the better spreadsheet even lets you specify the separator character), so when things are looking good in the Serial Plotter, you can use the Serial Monitor to export many of these lines and copy paste them into a spreadsheet to make pretty graphs and do other post processing.

The list of data values is interpreted as y-values for the same x-value, so all the graphs update at the same time and speed. The plotter will choose a different colour for each graph, the user has no control over this. The only way to influence this a little is by changing the order of the data.

Labels

The Serial Plotter does have one "fancy" option: it lets you specify a label for each graph (**Figure 3**). There are two ways to do this:

- Before sending any data values, first send a list of labels with the labels separated by either a comma, a whitespace, or a tab character and end with a newline character. Therefore, the labels themselves cannot contain these characters. The data must be sent in the same order as the labels to match things up. Example:

```
Serial.println("temperature,humidity,pressure")
```

- Send labels and values as pairs and separate the two with a colon (`,`), e.g.

```
Serial.println("temperature:21,humidity:76,
pressure:1007")
```

Make sure to always send these pairs in the same order to prevent the plotter from mixing up the data for different graphs. A list of label-value pairs must again be separated by either a comma, a whitespace, or a tab character and end with a newline character.

Note that as soon as you send a text string, it will be considered a label. It is therefore quite easy to mess up the labels. Another way to do this is to forget delimiters, making for instance data end up in a label.

Quirks & limitations

1. The horizontal axis goes on forever, there is no way to restart the graph programmatically. The only way to do this is to close the Serial Plotter and then open it again. Sometimes it is even necessary to do this twice to get rid of stale labels and/or data.

2. It is not possible to plot a point at a (x,y) position of your choice; new points are always added at the end.

3. The vertical axis scales automatically, which seems nice, but can be very annoying, especially when there are outliers or glitches that force the scale up and make it hard or even impossible to see the much smaller data of interest. This can also happen when one data series is in the range of, say, $[-1,+1]$, and another in the range $[-100,+100]$. In that case the scale will be from -100 to $+100$.

Some people try to outsmart the plotter by adding something like `min:-100,max:100` to the list of data values in an attempt to stop autoscaling, but this only works when every value stays within these bounds (**Figure 4**).

4. For reasons only known to the developer(s) of the Serial Plotter, the minimum scale usually is from -6 to $+6$, but -5 to $+5$ can happen too.

Helper function

Below is a small helper function that you can insert in a sketch to make the Serial Plotter slightly more user-friendly while keeping the sketch readable at the same time:

```
void plot(String label, float value, bool last)
{
    Serial.print(label); // May be an empty string.
    if (label!="") Serial.print(":");
    Serial.print(value);
    if (last==false) Serial.print(",");
    else Serial.println();
}
```

If the string *label* is specified, this function will print a `label:value` pair. If label is an empty string, then it will only print the value. This allows it to output pure CSV-formatted data in the Serial Monitor, good for use with spreadsheets.

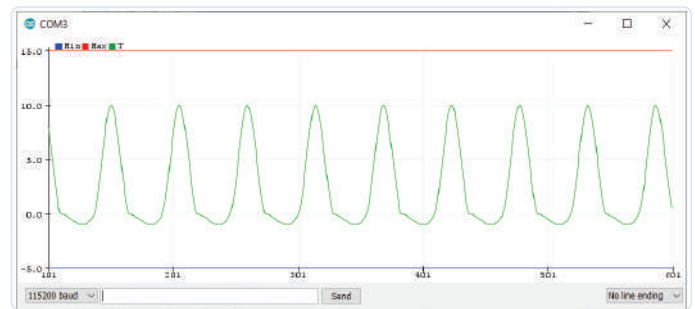



Figure 4: Sending constant minimum and maximum values avoids automatic scaling of the vertical axis, but only as long as all other values remain within these limits.

Note that you cannot have both the Serial Monitor and the Serial Plotter open at the same time.

After opening the serial port, call it like so:

```
// Graph1, more graphs follow.
plot("Temperature",temperature_value,false);
// Graph2, more graphs follow.
plot("Humidity",humidity_value,false);
// Graph3, last graph.
plot("Pressure",pressure_value,true);
```

Remember to restart the Serial Plotter (close and reopen it) every time you modify the sketch to ensure that all stale data and labels are removed. When in doubt, repeat this procedure. 

200540-01

Questions or Comments?

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

Contributors

Idea, Design and Text: **Clemens Valens**
Editor: **C. J. Abate**
Layout: **Giel Dols**

WEB LINK

- [1] **Source code of the Arduino IDE's Serial Plotter:**
<https://github.com/arduino/Arduino/blob/master/app/src/processing/app/SerialPlotter.java>

CLUE from Adafruit

A Smart Solution for IoT Projects

By **Tam Hanna** (Slovakia)

The BBC micro:bit was a great success, far beyond the educational market it was intended for. Now, Adafruit's CLUE has entered the fray with a fully-fledged display and far more memory. Complete with Bluetooth LE and a multitude of integrated sensors, it is especially suitable for smaller IoT projects.

Since the successes of Arduino and Raspberry Pi, it has become obvious that there is money to be made with educational computers of all kinds. In 2016, the BBC entered the race with the micro:bit, a single-board computer featuring a Bluetooth SoC from Nordic Semiconductor instead of a fully-fledged Linux-capable processor.

Since then, incredible stories have emerged of businesses who have built entire companies around the exclusive distribution of the micro:bit ecosystem [1].

The old saying that you "don't eat at a well-filled pot all by yourself for too long" also applies to the field of embedded computing. The continued progress in the field of Bluetooth SoCs has led to the

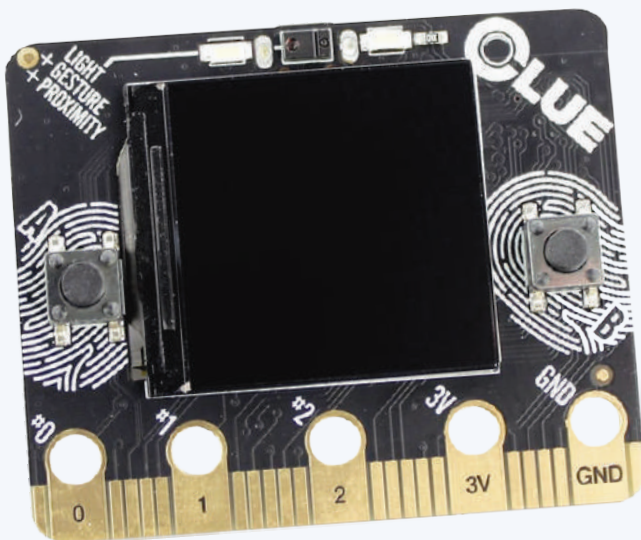


Figure 1: The attacker from the front ...

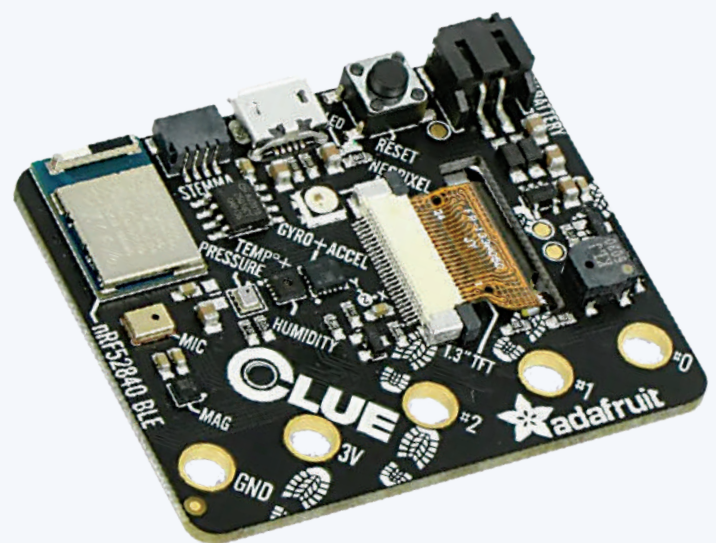


Figure 2: ... and from behind.

FEATURES

- › Nordic nRF52840 Bluetooth LE processor: 1 MB of flash, 256 kB RAM, 64 MHz Cortex M4 processor
- › 1.3" 240 × 240 color IPS TFT display for text and graphics
- › Power from any 3.6 V battery source (internal regulator and protection diodes)
- › Two user buttons and one reset button
- › Motion sensor, accelerometer/gyrometer, and magnetometer
- › Proximity, light, color, and gesture sensor
- › PDM microphone sound sensor
- › SHT humidity sensor
- › BMP280 sensor for temperature and barometric pressure/altitude
- › RGB NeoPixel indicator LED
- › 2 MB internal flash storage for data logging, images, fonts, or CircuitPython code
- › Buzzer/speaker for playing tones and beeps
- › Two bright white LEDs in front for illumination/color sensing
- › Qwiic/STEMMA QT connector for adding more sensors, motor controllers, or displays over I²C. GROVE I²C sensors also supported by using an adapter cable.
- › Programmable with Arduino IDE or CircuitPython

micro:bit's 16 MHz and 16 kB SRAM looking a little long in the tooth. Moreover, its 5 × 5 LED display is not suited to outputting anything more than the simplest of graphics.

Adafruit Attack

With the Nordic Semiconductor nRF52840 launch, a single-core Bluetooth SoC whose ARM processor reaches 64 MHz and

featuring 256 kB RAM — an attack vector has opened. **Figures 1 and 2** show the result — the Adafruit CLUE that looks very similar to the BBC micro:bit. Besides the SoC, which is not immediately visible in these photos, it is the much larger screen on the front that really stands out. Instead of LEDs, we're given a 240 × 240-pixel color display based on classic IPS LCD rather than organic technology.

Another nice touch of the module is the connector located on the back, as shown in **Figure 3**. It exposes an I²C bus using Adafruit's in-house format over which further sensors can be easily connected. There is also an adapter for the Grove format used by Seeed, from whom various reasonably-priced sensors are available.

It should be noted that the CLUE only looks to be partially compatible with its forebear. While the connector along the bottom edge is physically identical, using a different display means that, at first glance, it looked as though many of the available housings for the BBC micro:bit would not fit the Adafruit CLUE.

The author tested this hypothesis with a Thingiverse case from domw available at [2]. The front obviously did not fit as the CLUE's display was much larger than the LED matrix of the BBC's original. Bearing this in mind, the author found it particularly surprising that the rear panel of the case fitted correctly, despite the additional connectors. However, on closer inspection, this was likely because the case design was comparatively generous. Had the case been designed with a tighter fit, it is unlikely to have been any use at all.

A Question of Programming

Being an educational system, developing with a micro:bit is

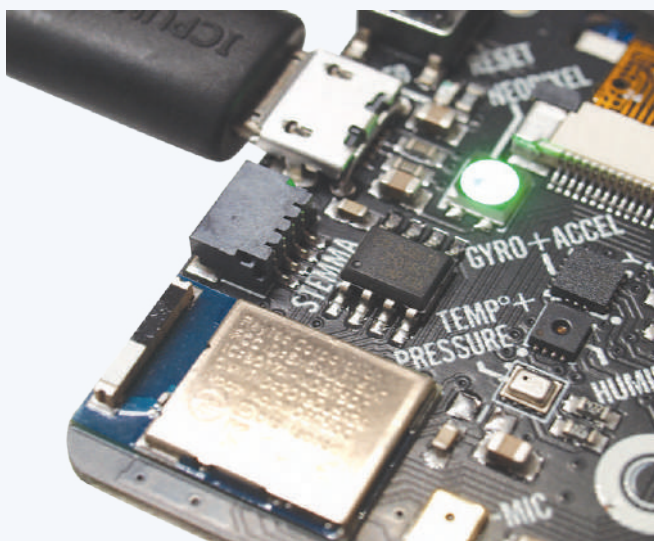


Figure 3: The STEMMA port allows the Adafruit CLUE to connect to extension boards.

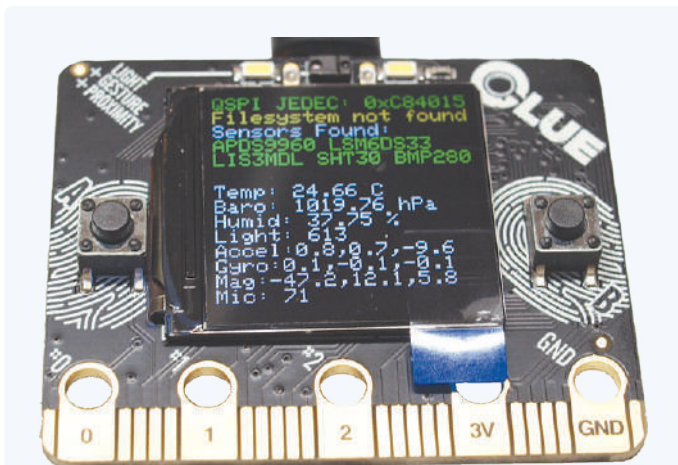


Figure 4: This pre-installed app shows the information returned by the sensors.



Figure 5: The Python runtime also exposes a virtual drive.

unlike the experience to be had using classic embedded development environments such as ARM Keil. This may be annoying for embedded purists, but it is necessary in practice as many universities do not have enough competent personnel for debugging C++ (believe the author: students create incredibly obtuse program errors).

Instead, it generally relies on a quadrumvirate of Arduino IDE, CircuitPython, MakeCode, and Scratch. For CLUE, however, only two of these environments are currently available. MakeCode is being worked on without a known delivery date, and there is no information on Scratch. What is included is a serial bootloader to enable the deployment of code, much like the Raspberry Pi Pico.

For a first, small experiment, let's get CircuitPython running. If you connect a new board to the computer via the Micro-USB connector on the back, the display shows a status page (**Figure 4**) that provides information on the operating state.

Pressing the reset button on the back of the board twice initially causes the frame buffer in the screen's display controller to freeze.

The connected workstation (the author runs Linux) then sees a new USB drive where compiled code can be uploaded.

Interestingly, the Adafruit CLUE is always visible to the computer. If it is not in bootloader mode, *dmesg* detects it as follows:

```
tamhan@TAMHAN18:~$ dmesg
...
[28292.202193] usb 1-2.7: Manufacturer: Adafruit LLC
[28292.202195] usb 1-2.7: SerialNumber: 7687A137B6FDB874
[28292.204040] cdc_acm 1-2.7:1.0: ttyACM0: USB ACM
device
```

After double-pressing the reset button, a USB drive appears instead, as shown here:

```
tamhan@TAMHAN18:~$ dmesg
...
[28371.624193] sd 10:0:0:0: Attached scsi generic sg6
type 0
```

It is important to note that this drive does not stay enabled forever. If it remains unused for more than 30 seconds or so, the firmware resets back to normal operation.

Search Files

By visiting the URL https://circuitpython.org/board/clue_nrf52840_express/, we can take our first step and download the *adafruit-circuitpython-clue_nrf52840_express-en_US-6.1.0.uf2* file. This contains the runtime that must be placed on the USB drive.

It is interesting to note that you will also find a file called *CURRENT.UF2* on the drive. This allows you to download the firmware that is currently in the memory of the target system.

Strangely, the runtime does not come with a full library that supports all the available sensors. Instead, we have to go to the URL <https://circuitpython.org/libraries> to download the archive *adafruit-circuitpython-bundle-6.x-mpy-20210329.zip* and then extract it to a convenient folder in the file system.

At this point, you should take another look at the CLUE's screen as the runtime permanently outputs the console's contents. A nice touch is that the device — as shown in **Figure 5** — exposes the internal memory of the Python working environment to the PC.

It is important to place the following folders from the archive in the *Libs* folder on the device:

```
adafruit_apds9960
adafruit_bus_device
adafruit_display_shapes
adafruit_display_text
adafruit_lsm6ds
adafruit_register
```

As if this were not enough work, Adafruit also expects you to collect the following individual files. Why these are not all bundled up in a single archive is unclear:

```
adafruit_bmp280.mpy
adafruit_clue.mpy
adafruit_lis3mdl.mpy
adafruit_sht31d.mpy
adafruit_slideshow.mpy
neopixel.mpy
```

Code Example

For a first simple attempt with the Python environment, you can use the example provided at <https://learn.adafruit.com/adafruit-clue/clue-spirit-level>. This implements a spirit-level application using several CLUE-specific idioms.

The first step of the code is to include a group of libraries:

```
import board
import displayio
from adafruit_display_shapes.circle import Circle
from adafruit_clue import clue
```

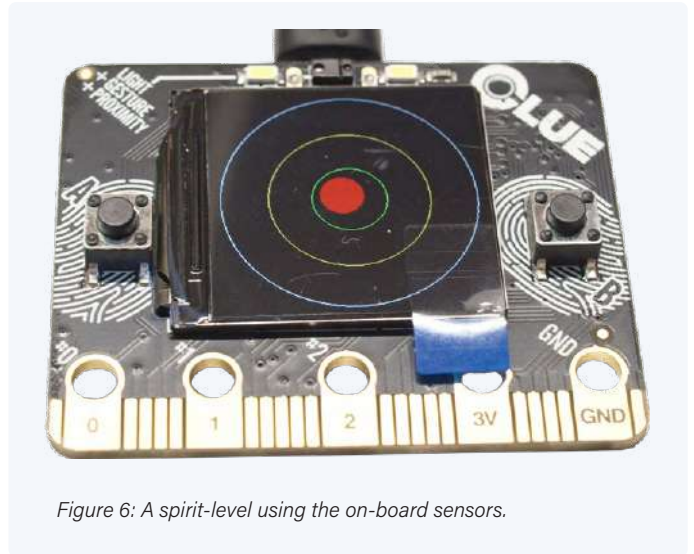


Figure 6: A spirit-level using the on-board sensors.

In addition to the `clue` object, which provides various board-related functions, the import of the `Circle` class is also interesting here. The GUI stack allows both direct drawing into a frame buffer

Advertisement

TAILORED TO YOUR NEEDS.

Custom & Standard Terminal Blocks



WÜRTH
ELEKTRONIK
MORE THAN
YOU EXPECT



Würth Elektronik Terminal Blocks

In addition to a portfolio of more than 2000 standard articles, Würth Elektronik offers various possibilities to tailor the products to your specific requirements. Personalized modifications of standard terminal blocks are available for small to medium quantities within a few days as a special service. Fully customized products in high quantities are possible within a few weeks. In house design, tooling and prototyping ensures all customer specific requirements are met.

For further information, please visit: www.we-online.com/TBL

- Highly customized products
- Over 2000 standard articles
- Available from stock without MOQ
- Fast delivery
- Personalized modifications of standard parts for small quantities
- Color & printing possibilities with MOQ for mass production

as well as working with objects that are converted by the firmware into elements that appear on the screen.

In the next section, the firmware initializes a reference to the display and assembles a display group object:

```
display = board.DISPLAY
clue_group = displayio.Group(max_size=4)
```

The `clue_group` object is interesting in that it generates a parent element reminiscent of a DOM tree. Our code then more or less writes arbitrary objects to this tree for display.

Looking at the image in **Figure 6**, it follows that the next step of the program is to generate the three circles responsible for the representation of the deflection and to register them for the output:

```
outer_circle = Circle(120, 120, 119, outline=clue.WHITE)
middle_circle = Circle(120, 120, 75, outline=clue.
    YELLOW)
inner_circle = Circle(120, 120, 35, outline=clue.GREEN)
clue_group.append(outer_circle)
clue_group.append(middle_circle)
clue_group.append(inner_circle)
```

Next are some more housekeeping tasks, the sense of which is most easily understood by examining the sample code below:

```
x, y, _ = clue.acceleration
bubble_group = displayio.Group(max_size=1)
level_bubble = Circle(int(x + 120), int(y + 120), 20,
    fill=clue.RED, outline=clue.RED)
bubble_group.append(level_bubble)

clue_group.append(bubble_group)
display.show(clue_group)
```

Last but not least, we need a loop that analyzes the position values output by the Adafruit library via the `acceleration` attribute and writes them to the coordinate properties of the `bubble_group` object:

```
while True:
    x, y, _ = clue.acceleration
    bubble_group.x = int(x * -10)
    bubble_group.y = int(y * -10)
```

The most convenient way to quickly execute code on the CLUE is to use the `code.py` file shown in Figure 5. The CircuitPython firmware automatically executes this as part of each startup. Figure 6 shows what you can expect.

Due to the presence of a Bluetooth radio, this can also be used to communicate with the host computer. At [3], Adafruit provides an entertaining example that illustrates the use of the web Bluetooth API implemented in Google Chrome.

And Now Using C

Python may be a fast way to get unbureaucratic results from an embedded system. However, maximum performance is achieved by using C.

Especially on a single-core radio system, implementing communication is challenging if the application is not to be beset by timing problems. Because of this, Adafruit more or less forces developers to use the Arduino IDE. A real-time operating system then works in the background, allocating computing power to the various tasks.

Under Linux, the first step requires an extension package that allows the Arduino IDE (version 1.8.6 or higher) to communicate with the CLUE's non-standard bootloader:

```
tamhan@TAMHAN18:~$ pip3 install --user adafruit-nrfutil
Collecting adafruit-nrfutil
...
Successfully installed adafruit-nrfutil-0.5.3.post13
```

Next, we need to enter the URL https://www.adafruit.com/package_adafruit_index.json in the Board Manager to make the Adafruit nRF52 board package available for download. After these steps are complete, the board is available under **Tools > Board > Adafruit CLUE**.

Unfortunately, as in the case of CircuitPython, setting up these libraries and other settings is a laborious process. More information on this can be found at [4].

Is It Worth It?

If you do decide on the CLUE, you're getting a thoroughly attractive evaluation platform that is pleasant to use in the area of interfacing coupled with the benefits of a color screen. On the other hand, the price is comparatively high compared to the BBC micro:bit, which costs considerably less. It should also be noted that the CLUE is not 100% compatible with the micro:bit. Experience tells us that

WEB LINKS

- [1] StreamIT, s. r. o. : <https://edutronik.sk/v2/>
- [2] Housing: <https://www.thingiverse.com/thing:1767446>
- [3] Demo application: <https://learn.adafruit.com/bluefruit-dashboard-web-bluetooth-chrome>
- [4] Info: <https://learn.adafruit.com/adafruit-clue?view=all>

this not insignificant detail has the greatest impact precisely when we are in difficulty, such as porting an existing, working project between the two platforms.

For those who want to work with a clean, Nordic-based microcontroller or radio module, you'll probably be better served by a classic evaluation board. So the bottom line is that the CLUE is an endearing product for those who want to benefit from a BBC micro:bit but require a little more performance or use of a full display. ◀

210395-01

Questions or Comments?

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

Contributors

Text and Photographs: Tam Hanna
Editor: Jens Nickel
Translation: Stuart Cording
Layout: Giel Dols



RELATED PRODUCTS

➤ Adafruit CLUE
www.elektor.com/19512

Advertisement

The advertisement features a background collage of numerous Elektor magazine covers. Overlaid on this are several key elements: a large black banner with the text 'Elektor Archive' and '1974-2021' in a white, typewriter-style font; a red circular badge in the upper right corner stating 'NEW Now incl. volume 2021!'; a blue rectangular box in the lower left with the text 'More information: www.elektor.com/20071'; a QR code in the center; and two physical Elektor products, a square metal case and a black rectangular box with a gold 'elektor MAG' label. The Elektor logo, consisting of a stylized 'e' in a red circle followed by the word 'elektor' and the tagline 'design > share > earn', is positioned in the bottom right corner.

Buffer Board

for the Raspberry Pi 400

Protect the I/Os



By Ton Giesberts (Elektor)

Released in November 2020, the Raspberry Pi 400 is actually a Raspberry 4, but built into a keyboard, with the GPIO expansion connector on the back of the enclosure. Connecting additional hardware always brings risks, especially during prototyping. The buffer board we present here is especially designed for the Raspberry Pi 400. It enables interfacing with both 3.3 V and 5 V external logic for all the 26 GPIOs, and the buffers/level shifters used for this also provide ESD protection.

Raspberry Pi boards will hardly need any introduction here. Since the introduction of the first version in 2012, they have been the subject (or part) of many projects in *Elektor* magazine. We have presented quite a few different hardware projects and extension PCBs for these processor boards. For this project, let's spoil the suspense right away: the circuit I will cover is not new, as its first version was discussed back in 2015 an Elektor Labs page [1]. In 2018, the design was adapted to the expanded, 40-pin I/O connector that has been the standard for connecting additional hardware to these processor boards since the introduction of the Raspberry Pi B+ [2]. And now, the latter version is adapted to one of the more recent members of the Raspberry Pi product family — the Raspberry Pi 400.

Very simply put, the 400 is a Raspberry 4, built into a keyboard. It resembles the once-famous computers of the 1980s, like the Commodore 64, Sinclair Spectrum, Acorn BBC and similar. In terms of specifications, the Raspberry Pi is of course many times more powerful than its now antique predecessors, but there is also a striking similarity — the GPIO expansion connector on the back of the enclosure, to which the user can connect external (self-designed) hardware. Connecting additional hardware always brings risks, especially during prototyping, and the buffer board presented here prevents the Raspberry Pi from being damaged in the process. In addition, the buffer board enables interfacing with both 3.3 V and 5 V external logic, and the buffers/level shifters used for this also provide ESD protection.

Hardware

The project's schematic (Figure 1) is exactly the same as the one from our publication in 2018. The TXS0108E 8-bit buffers/voltage translators used in this project are bidirectional, and each A-port and B-port pin has a pull-up resistor. The pull-down resistor of a GPIO of the Raspberry Pi is typically in the order of 40 to 60 k Ω . This value is too high to properly pull-down the I/O when the buffer board is plugged in. So, please note that with this input configuration the logic level will not be low; the pull-down will not work as intended. The I/Os have separate power supply pins for the Raspberry Pi side and the outside-world, VCCA and VCCB, respectively. Every A-port I/O of the TXS0108E has a pull-up resistor to VCCA, connected to the +3.3 V power supply of the Raspberry Pi 400, and each B-port I/O has a pull-up resistor to VCCB. VCCB — for the level of the I/Os on K2 — can be set to either +3.3 V or +5 V logic by jumper JP3. The pull-ups of the buffers have a value of 40 k Ω when the output is driving low and a value of 4 k Ω when the output is driving high. So the outputs of the buffers are in fact open drain. If, for instance, an LED is connected from the output of the buffer to ground, a voltage divider is created when an extra series resistor is used. A resistive load on the output will cause the logic high level to drop. Something to keep in mind!

There are two 0.5 A PPTC resettable fuses (F1 and F2) in the power connections between K1 to K2 on the buffer board to protect the +5 V and +3.3 V power supplies of the Raspberry Pi 400.

For the implementation of an I²C bus to communicate with external hardware, GPIO2 is SDA and GPIO3 is SCL, extra pull-up resistors R1 and R2 can be enabled by jumpers JP1 and JP2.

During boot of the Raspberry Pi GPIO0 (ID_SD) and GPIO1 (ID_SC) are used to read an EEPROM of an I²C HAT (Hardware Attached on Top). After boot-up these GPIO's can be used like the 26 others, but

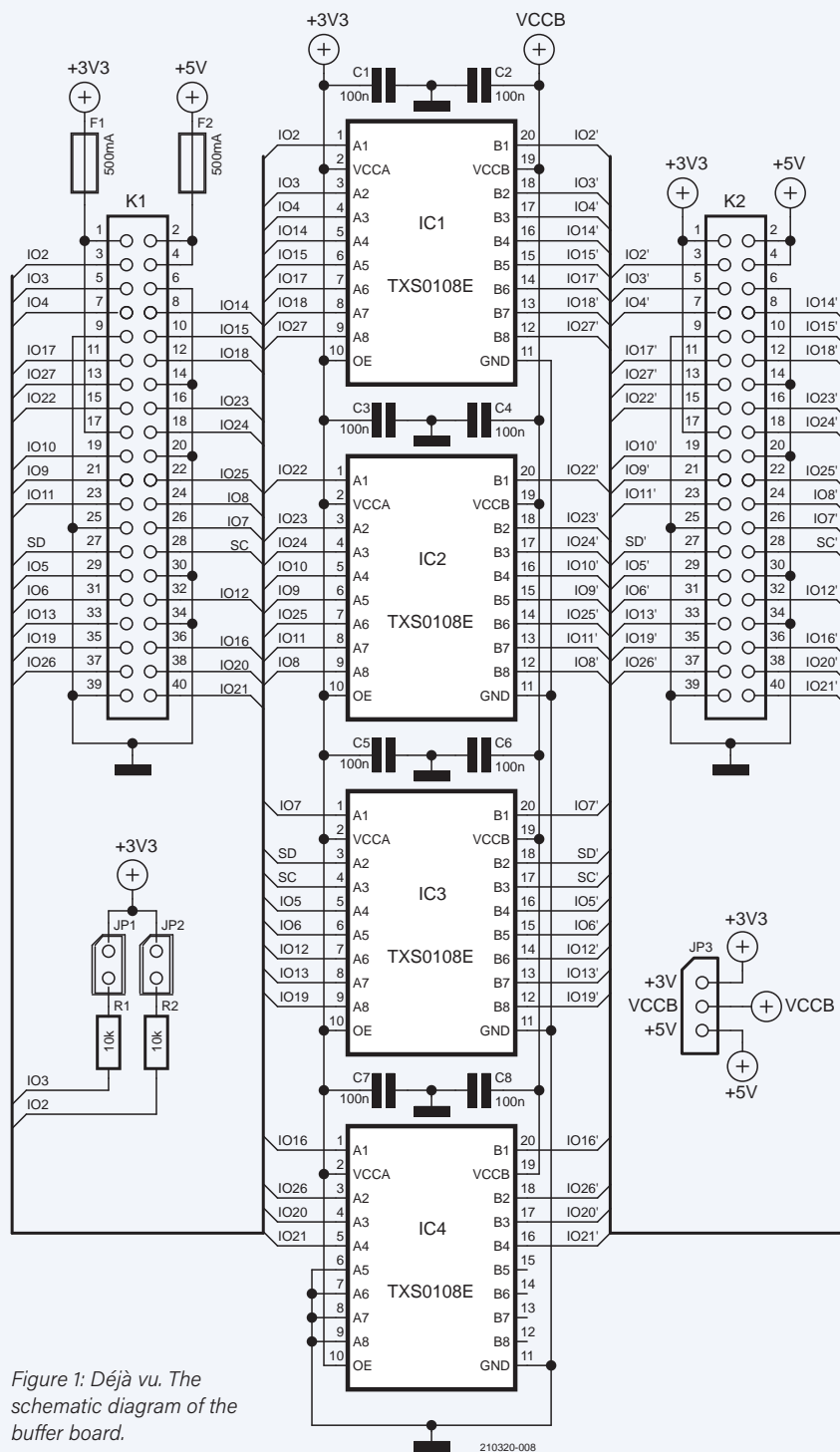


Figure 1: Déjà vu. The schematic diagram of the buffer board.

care needs to be taken if an I²C HAT is mounted the system isn't affected. To prevent the reading of GPIO0 and GPIO1 during boot add the following entry to `/boot/config.txt`:

```
force_eeprom_read=0
```

For more information, look at the Raspberry Pi documentation about the `config.txt` file [3].



COMPONENT LIST

Resistors

R1,R2 = 10 k Ω , 100 mW, 1 %, SMD 0603

Capacitors

C1...C8 = 100 nF, 50 V, 10 %, X7R, SMD 0603

Semiconductors

IC1...IC4 = TXS0108EPWR, SMD TSSOP-20

Others

K1 = 2 x 20 receptacle, right angle, pitch 2.54 mm

K2 = 2 x 20 pin header, vertical, pitch 2.54 mm

JP1,JP2 = 2-way pin header, vertical, pitch 2.54 mm

JP3 = 3-way pin header, vertical, pitch 2.54 mm

JP1,JP2,JP3 = shunt jumper, 2.54 mm spacing

F1,F2 = PPTC resettable fuse, SMD, polyfuse, 1210L050YR Littelfuse

PCB 210320-1 v1.0

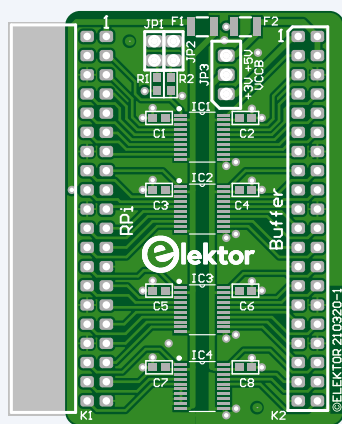


Figure 2: The layout of the new buffer PCB.

The Printed Circuit Board

The schematic may be old news, but the PCB (**Figure 2**) is adapted especially for the Raspberry Pi 400. It's a bit smaller than the original buffer board presented in 2018. The Gerber files for this new board are available for download, so you can order it at the PCB manufacturer of your choice. But it is, of course, a lot more convenient to buy the completely assembled buffer board in the Elektor Store [4].

A right-angled receptacle is used for the connector on the Raspberry Pi 400 side of the buffer board (K1) so it can be inserted in the GPIO header in the back of the enclosure (**Figure 3**). The connector for the buffered I/Os is a standard 40-pin male vertical header (K2). The size of the module is 55 x 44 mm, including receptacle K1, which protrudes over the edge of the PCB. Compared to the original 150719-1 PCB, the two rows of pins of K1 are swapped, because a receptacle is used here. Placing a standard vertical male 40-way

pin header for K1 to connect this buffer board to a Raspberry Pi 2, 3 or 4 via a flatcable — like with the older version of the buffer — will not work here. However, **Figure 4** shows that this module can still be used with a Raspberry Pi 2, 3, or 4.

The output connector K2 can be connected to external circuits using a short 40-way ribbon cable with two 2 x 20 receptacle connectors attached, or just a single receptacle with short wires soldered to them or single sockets with wires. But be careful pressing a 40-way receptacle onto K2 or unplugging it from the board. Don't do this while the buffer board is still inserted in the Raspberry Pi 400, because quite some force is needed and the Raspberry Pi 400's GPIO header could get damaged.

Testing the Buffer Board

Two very simple Python programs for testing the buffer board – borrowed from the older project – are available for download at the Elektor Labs page of this project [5]. One is to test all GPIOs as output, *Check_all_GPIOs_as_output.py*, and the other is to test all GPIOs as input, *Check_all_GPIOs_as_input.py* (210320-11.zip). In Raspbian, just double click on one of the files and the default IDE for Python will open, then select RUN to start the test.

When testing the GPIOs as output, only a single low-current LED connected between a pin and GND is needed to see if an output is working or not. As a series resistor for the LED a 1.8 k Ω resistor can be used, but the value is not really critical. It will limit the current through the LED if it is directly connected to the positive supply voltage. Outputs are sequentially tested in four groups of maximum eight pins each, named IOA to IOD. Because of the open drain output the voltage across an LED (red) plus resistor is about 2.6 V, when 5 V is selected as a power supply for the outputs (JP3). Connect the resistor plus LED to one of the selected outputs and it will be switched on for 0.2 seconds. The repetition rate of this pulse depends on the size of the group: 1.6 seconds for groups A to C (each with eight outputs) and only 0.4 seconds for group D (two outputs). Change 'IOA' to one of the other groups in the line

```
for i in IOA:          # leds blink 0.2 s in IOx group
```

to test the other groups of outputs. Of course, GPIO0 and GPIO1 (ID_SD and ID_SC) can be added to one of the groups too.

The program to test the GPIOs as input uses one I/O as output to indicate that the input under test is working, GPIO3 by default. Connect a 1.8 k Ω resistor and LED between pin 5 (IO3') of K2 and GND. Only one input at a time is tested in the source code, to make sure only this one is working as input. Change the number in the following line to test another GPIO as input:

```
IN1 = 2  #selected GPIO to test as input
```

The program also prints the selected GPIO and its input level. The inputs have their pull-ups enabled. So to make the connected LED

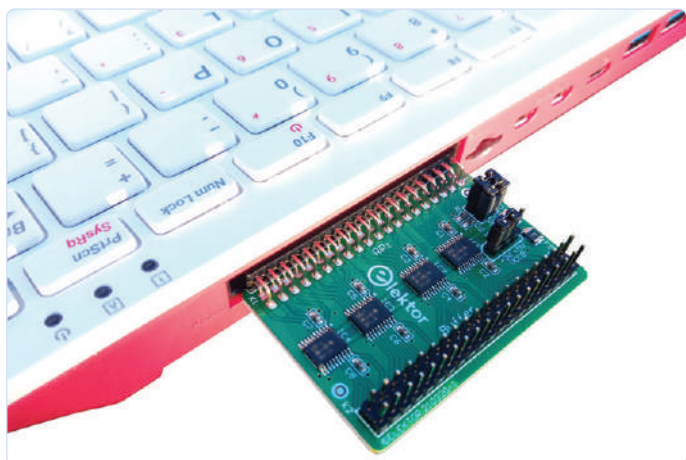


Figure 3: The buffer board plugged into the Raspberry Pi 400.

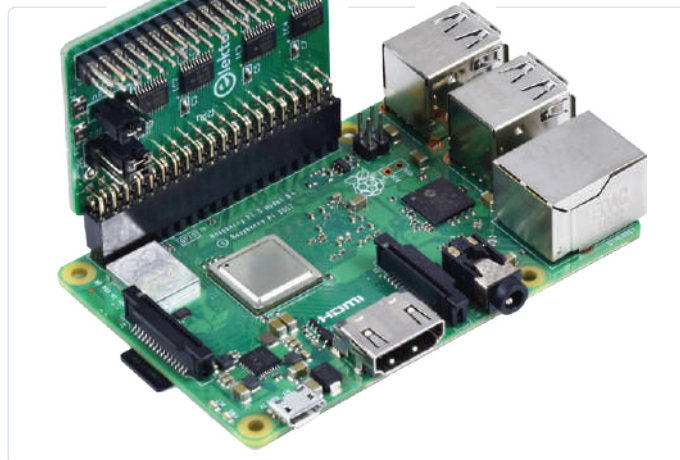


Figure 4: The board can also be used with 'classic' Raspberry Pi boards.

light the current input pin must be connected to ground. When done so the printout will change. Finally, select a different GPIO for the output so you can also check GPIO3 as input. Of course, there are numerous ways to test the GPIOs, if anyone has a more efficient and/or faster way: please share.

This buffer allows you to connect new hardware to the Raspberry Pi 400 with peace of mind, greatly reducing the chance of it getting damaged during experiments. But still, reducing the risk by using this buffer board doesn't mean that nothing can go wrong. In many cases, I won't hurt to add some common sense too. ◀

210320-01

Contributors

Design, text and illustrations: Ton Giesberts

Editor: Luc Lemmens

Layout: Giel Dols

Questions or Comments?

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



RELATED PRODUCTS

- **Raspberry Pi 400 Buffer Board (SKU 19884)**
www.elektor.com/19884
- **Raspberry Pi 400 Kit – Raspberry Pi 4-based PC Kit (EU) + Free GPIO Header (SKU 19431)**
www.elektor.com/19431
- **Raspberry Pi 400 – Raspberry Pi 4-based PC Kit (US) + Free GPIO Header (SKU 19429)**
www.elektor.com/19429

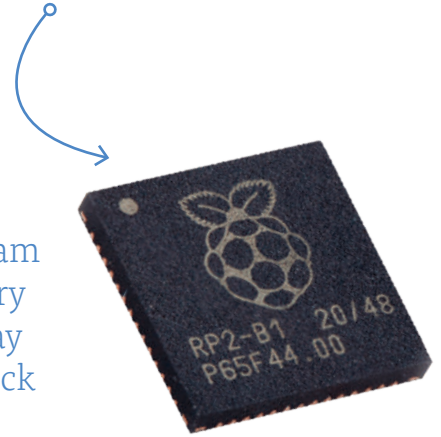
WEB LINKS

- [1] Raspi Buffer Board on Elektor Labs: <https://www.elektormagazine.com/labs/raspi-buffer-board>
- [2] Guy Weiler: "Raspberry Pi Buffer Board," Elektor 11-12/2018: <https://www.elektormagazine.com/magazine/elektor-52/42084>
- [3] More on config.txt: https://www.raspberrypi.com/documentation/computers/config_txt.html
- [4] Raspberry Pi 400 Buffer Board in the Elektor Store: <https://www.elektor.com/raspberry-pi-400-buffer-board>
- [5] This project's page on Elektor Labs: <https://bit.ly/3GdRJ4G>

Raspberry Pi RP2040 Boards Aplenty

By **Mathias Claußen** (Elektor)

The RP2040 is the first microcontroller chip designed by the team at Raspberry Pi. It was first fitted to the maker-friendly Raspberry Pi Pico board, and since its introduction, it has also found its way onto boards and kits from third-party providers. It's time to check out what they have to offer!



The RP2040 is certainly an interesting alternative to the more established microcontrollers. Not only is the chip's bang-per-buck ratio impressive, it currently enjoys a good level of availability. Documentation and support from the Raspberry Pi Foundation is another of its strong points, making it a good choice for newbies to the environment. All the latest information relating to this chip can be read and viewed in articles [1][2][3], a webinar [4] or videos [5] from Elektor.

The Raspberry Pi Pico — the manufacturer's own board on which the RP2040 is installed — is equipped with minimal additional hardware to keep the price down to around €5 per board. One year on after its release, the RP2040 chip has found its way onto a number of third-party boards which have been equipped with a wide variety of peripherals. In this review we take a closer look at some of these other boards to help you to decide which of them fits your needs.

The Raspberry Pi Pico

The Raspberry Pi Pico board (**Figure 1**) contains just about the

minimum hardware necessary to support the RP2040 operation. On board is a user-controllable green LED and a DC/DC buck/boost converter to allow the board to be powered from an external 1.8 to 5.5 V source or via the 5 V from the USB port. The Raspberry Pi Pico is still one of the best boards in terms of price/performance ratio, especially if you just want some experience with the RP2040 and its development environment. Maybe you already have a collection of modules and external components/sensors you can interface with it. The board enjoys support from a number of books and there are many web resources to draw on for self-study. Additional hardware will be necessary if you plan to venture beyond the basics (see the Elektor Raspberry Pi Pico Experimenting Kit below). Without too much effort the Pico board can also function as a debugger for another RP2040. At €5 per unit, it is very affordable, and if you want a Pico with preinstalled pin headers together with a suitable micro USB cable (**Figure 2**), look no further than the Elektor Store, where you can also find some interesting expansion boards for the Raspberry Pi Pico there.

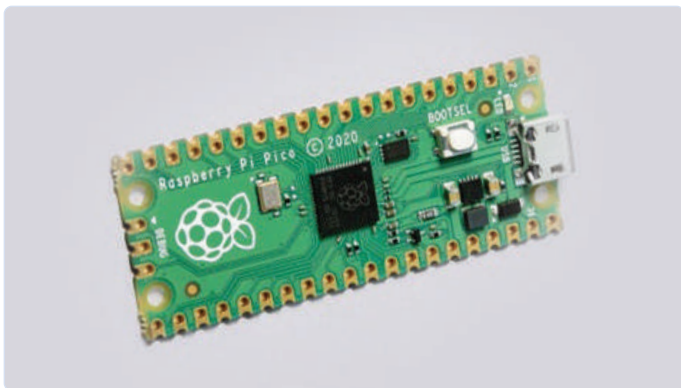


Figure 1: The Raspberry Pi Pico (Source: Raspberry Pi).

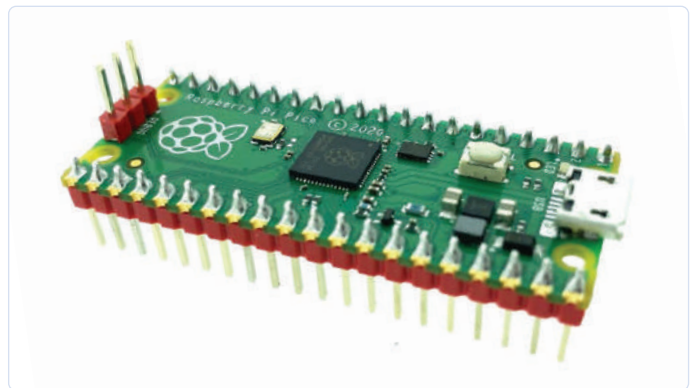


Figure 2: Raspberry Pi Pico with Headers (Source: Elektor).

Adafruit Feather RP2040

The Adafruit Feather RP2040 (**Figure 3**) is an RP2040 board in the Feather board form factor. The difference to the Raspberry Pi Pico is mainly the number of peripherals and the amount of Flash memory integrated on board. The Raspberry Pi Pico has just 2 MB, while the Feather Board has 8 MB SPI Flash, giving significantly more space for your own software. An RGB LED is fitted to the board, as well as a USB-C socket and a STEMMA QT connector for plugging in any Qwiic, STEMMA QT or Grove I2C device. The JST PH battery connector allows the board to be powered by a single cell LiPoly battery which can be charged directly via the on board USB port.

SparkFun Thing Plus - RP2040

The SparkFun Thing Plus - RP2040 (**Figure 4**) is very similar to the Adafruit Feather RP2040, even down to the pin out assignments. This board is fitted with 16 MB QSPI Flash (underneath the board) which is the maximum addressable by the RP2040. It is also fitted with an RGB LED and three status LEDs. As with the Adafruit Feather RP2040, it includes a charging circuit for a single cell lithium battery and a Qwiic connector. A micro SD card slot is also fitted underneath the board and SD cards can be addressed as mass storage using the RP2040's PIO state-machine facility. A webinar [4] reveals how the flexibility of the PIO state machines led to an unusual sequence of GPIO assignments to Sparkfun's SD card reader interface. Anyone with peripherals which use the Qwiic connector or add-ons with a feather board pinout should take a closer look at this board!

Arduino Nano RP2040 Connect

The Arduino Nano RP2040 Connect (**Figure 5**) brings Wi-Fi and Bluetooth communication capability to the RP2040 MCU. At €27, it is certainly not the cheapest board, but it is well equipped with just about every on board peripheral you could wish for. It is fitted with 264 KB of SRAM and 16 MB flash and has a ublox NINA-W102 Wi-Fi and BLE v4.2 module, a 6-axis IMU (STM LSM6DSOXTR), a microphone (MP34DT05) and an ATECC608A crypto Chip from Microchip. It is of course well supported by the Arduino IDE and despite all the additional peripherals, at 43.18 x 17.78 mm it is substantially smaller than the Raspberry Pi Pico board.

There is more than enough hardware on board to take the first steps with an RP2040 and to support even more ambitious projects. The possibility of exchanging data via Bluetooth and Wi-Fi, as well as the integrated microphone, also suggest an introduction to the first AI application. I am not sure I would recommend the Arduino Nano RP2040 Connect for a complete beginner to this environment. If your plan is to start working with Wi-Fi and IoT applications, you might be better advised to look at an ESP32 board or the new ESP32-C3. The datasheet for NINA-W102 module fitted to the Arduino Nano RP2040 Connect indicates that there is in fact an ESP32 tucked away inside to take care of Wi-Fi and Bluetooth communications. Those with some experience working with Wi-Fi and Bluetooth and are keen to try out an RP2040 with a cloud connection might want to check out what this board has to offer. My colleague Clemens Valens has already posted a short video about it, which you can find on the Elektor Youtube channel [6].

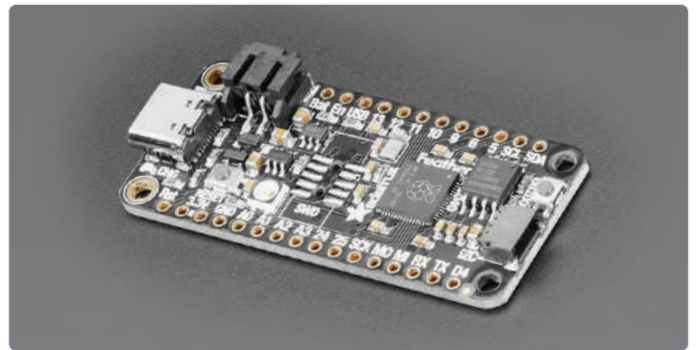


Figure 3: The Adafruit Feather RP2040 (Source: Adafruit).

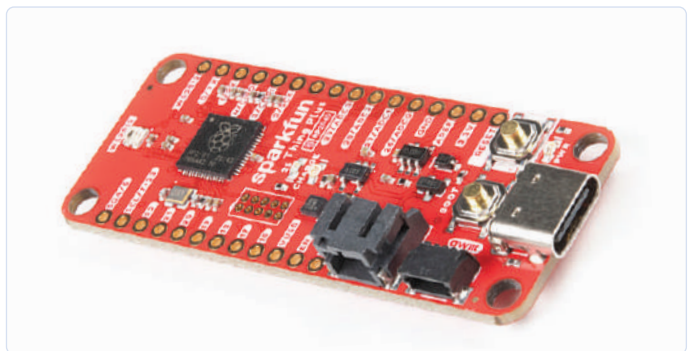


Figure 4: SparkFun Thing Plus RP2040 (Source SparkFun).

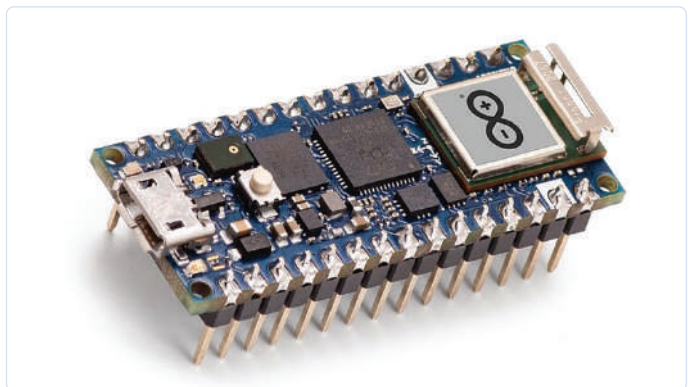


Figure 5: Arduino Nano RP2040 Connect (Source: Arduino.cc).

Cytron Maker Pi Pico (with soldered Raspberry Pi Pico)

If you want to start experimenting with the Raspberry Pi Pico and the RP2040, this platform (**Figure 6**) is a good place to start. The version which uses a soldered-in RPi Pico board retails for around €18.

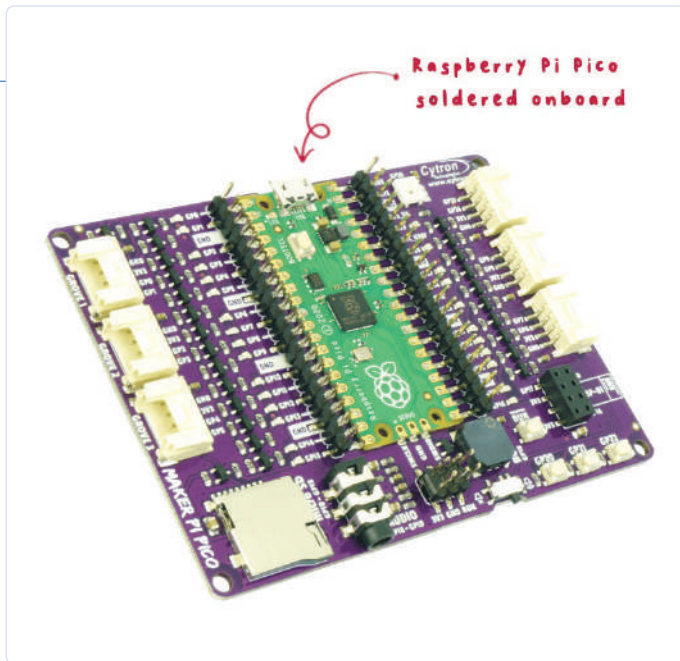


Figure 6: Cytron Maker Pi Pico (Source: cytron.io).

The main difference between this board and the others mentioned so far is its format. The Cytron Maker Pi Pico Base is an experimental platform onto which a Raspberry Pi Pico is fitted. All pins of the Raspberry Pi Pico are brought out onto two rows of pin headers, so that they can easily be probed with a measuring device. The board also offers a number of Grove connections to hook up compatible peripherals. Each GPIO pin on the board has an LED associated with it, so you can quickly see their state. A micro SD slot and a socket for an ESP-01 Wi-Fi board are also fitted along with a number of push buttons and a buzzer.

This platform is ideal if you already have a stack of Grove-compatible hardware; you can just plug them in and start working with the RPi Pico board. The pin headers also allow other modules to be connected quickly and safely. The mounted peripherals include a buzzer (with a disable switch), WS2812 RGB-LED, buttons and an SD card slot, with

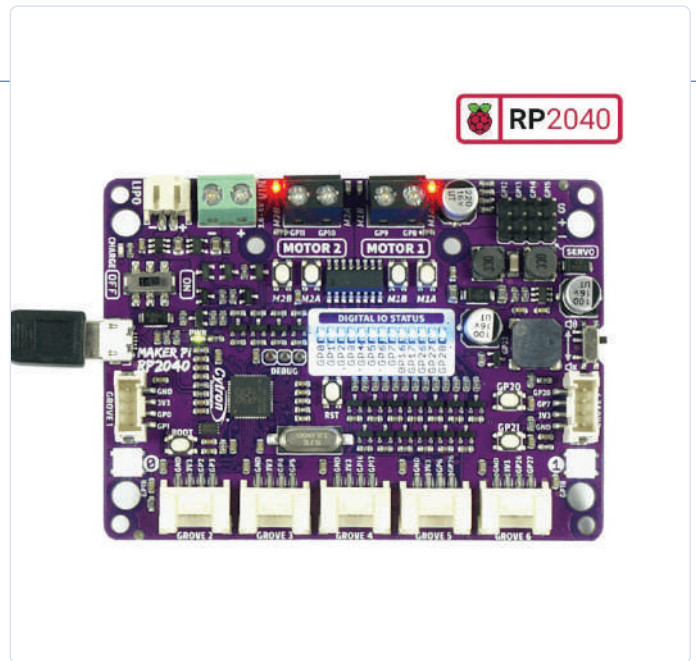


Figure 7: Cytron Maker Pi RP2040 (Source: cytron.io).

which you can make a start on increasingly ambitious pico projects step by step. The ESP-01 socket header also gives the opportunity to quickly configure a Wi-Fi project. The Raspberry Pi Pico's 3-pin debug interface is also available at header pins on the board.

Cytron Maker Pi RP2040

Motor control and robotics? With the Raspberry Pi RP2040? Sometimes it would be useful to be able to control small motors or a stepper motor with the Raspberry Pi Pico. This is exactly the aim of the Cytron Maker Pi RP2040 (Figure 7). The board contains an integrated (MX1508/TC1508) motor driver chip with two H-bridges to drive two low voltage DC motors or one stepper motor. If you are unsure how you can use an H-bridge or how motor control is accomplished, take a look at my basic article [7].

Small motors rated up to 6 V and 1 A per channel can be operated directly from this board. The board also offers the option of connecting up to four servos directly. Power for the board comes from the USB port, a LiPo battery or an external 3.6 – 6 V supply. A LiPo charging circuit is included for battery management and power from all of these sources can be turned off with an on-board switch.

The RP2040 chip is mounted on the Cytron Maker Pi RP2040 board and offers exactly the same Flash memory capacity (2 MB) as the RPi Pico board. The board also includes 13 GPIO status LEDs, two WS2812 LEDs, a buzzer, two buttons and seven Grove ports for easy expansion.

Elektor Raspberry Pi Pico Experimenting Kit

At € 45 the Elektor Raspberry Pi Pico Experimenting Kit (Figure 8) is the most expensive board in the line-up here. The basis for this kit is the Raspberry Pi Pico which plugs directly into this board so that it can easily swapped out if necessary. The main board of the kit itself provides buttons, LEDs, buzzers, a TFT display and Grove connectors. With the accessories of WS2812 LEDs, DHT11 air temperature and humidity sensor, relay, potentiometer, ultrasonic distance sensor, servo, MPU6050 gyro

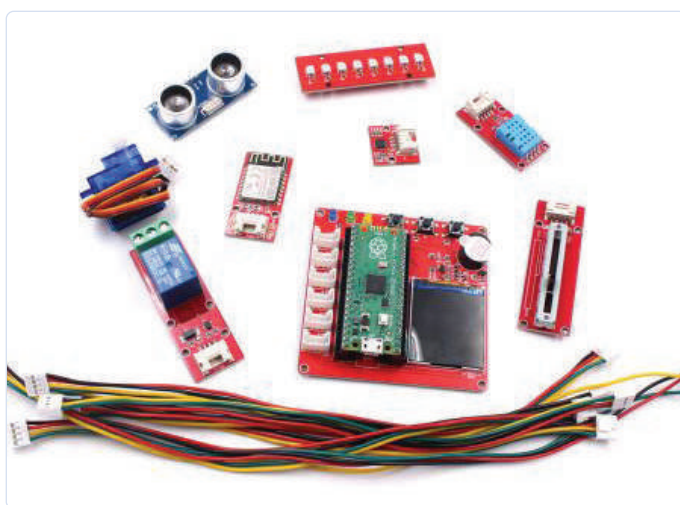



Figure 8: Elektor Raspberry Pi Pico Experimenting Kit (Source: Makerfabs).

and acceleration sensor as well as an ESP8266, you get a kit that is equally suited to beginners and advanced users who like to experiment. Since the components are easily plugged into the board as modules (via the Grove connectors), it provides real flexibility in how the Raspberry Pi Pico board is used. Thanks to the 1.44 inch display (with an ST7735 controller), your first graphics applications can also be developed using Python or C/C++. If you don't have any modules or other components already, this kit represents good value for money and altogether is a very well-rounded introduction to the world of the RP2040.

Which Board Is Best?

This is a question everyone has to answer for themselves and depends heavily on what they plan to do. Every board or kit has its advantages and disadvantages, and depending on your budget, you have to weigh up what the most essential features are for your needs. Do you need a board with all the peripherals already installed, or would you be better off starting with a basic kit and then adding specific modules as needed? Will your application require a battery management/charging function? Will the board be for your own use, or maybe you are thinking it would make a good gift to inspire a member of the upcoming generation of Engineers or Makers?

It's clear we can make no general recommendation for any particular board. What is clear is that the Raspberry Pi RP2040 microcontroller offers lots of potential for both beginners and experienced developers - and, above all, is currently in stock 

210629-01

Questions or Comments?

If you have any technical questions or comments about this article contact the author at mathias.claussen@elektor.com or contact Elektor at editor@elektor.com.

Contributors

Development and Text: **Mathias Claußen**
Editor: **Jens Nickel**
Layout: **Giel Dols**



RELATED PRODUCTS

- **Elektor Raspberry Pi Pico Experimenting Kit (SKU 19834)**
www.elektor.com/19834
- **Cytron Maker Pi RP2040 - Robotics with Raspberry Pi RP2040 (SKU 19926)**
www.elektor.com/19926
- **Cytron Maker Pi Pico (SKU 19706)**
www.elektor.com/19706
- **Arduino Nano RP2040 Connect with Headers (SKU 19754)**
www.elektor.com/19754
- **SparkFun Thing Plus - RP2040 (SKU 19628)**
www.elektor.com/19628
- **Adafruit Feather RP2040 (SKU 19689)**
www.elektor.com/19689
- **Raspberry Pi Pico RP2040 (SKU 19562)**
www.elektor.com/19562
- **Raspberry Pi Pico RP2040 (with pre-soldered Headers) (SKU 19568)**
www.elektor.com/19568

WEB LINKS

- [1] "Pico Power: Get to Know the Raspberry Pi Pico Board and RP2040," Elektormagazine.com: www.elektormagazine.com/articles/pico-power-raspberry-pi-pico-rp2040
- [2] "Raspberry Silicon: Introducing the Raspberry Pi RP2040 MCU and the Pico Board," Elektormagazine.com: www.elektormagazine.com/news/introducing-raspberry-pi-rp2040-microcontroller-pico-board
- [3] Raspberry Pi Pico MCU with Preinstalled Pin Headers: www.elektormagazine.com/news/raspberry-pi-pico-mcu-pre-installed-pin-headers
- [4] Eben Upton and Nathan Seidle discuss the Raspberry Pi Pico and RP2040: www.elektormagazine.com/news/eben-upton-nathan-seidle-sparkfun-raspberry-pi-pico-rp2040
- [5] Elektor TV on YouTube: www.youtube.com/user/ElektorIM
- [6] Clemens Valens, "Board Review: Arduino Nano RP2040 Connect," ElektorTV: www.youtube.com/watch?v=2EnCf64zZSA
- [7] Mathias Claußen, "Driving Motors with H-Bridges," Elektor January/February 2022: www.elektormagazine.com/210491-01

A Handbook on DIY Electronic Security and Espionage

SRAM Heated or Deep-Frozen



By **Luka Matic** (Croatia)

Here we demonstrate the hacking into static RAM (SRAM) data and encrypting it by off-the-record methods, basically stretching the device's data retention abilities. Having digested the theory, you're set to think of your own variants or ways to enhance and automate what's shown here. Do try this at home, please!

Editor's Note. *This article is an excerpt from the 224-page A Handbook on DIY Electronic Security and Espionage. The excerpt was formatted and lightly edited to match Elektor Mag's editorial standards and page layout. Being an extract from a larger publication, some terms in this article may refer to discussions elsewhere in the book. The Author and Editor have done their best to preclude such instances and are happy to help with queries. Contact details are in the **Questions or Comments?** box.*

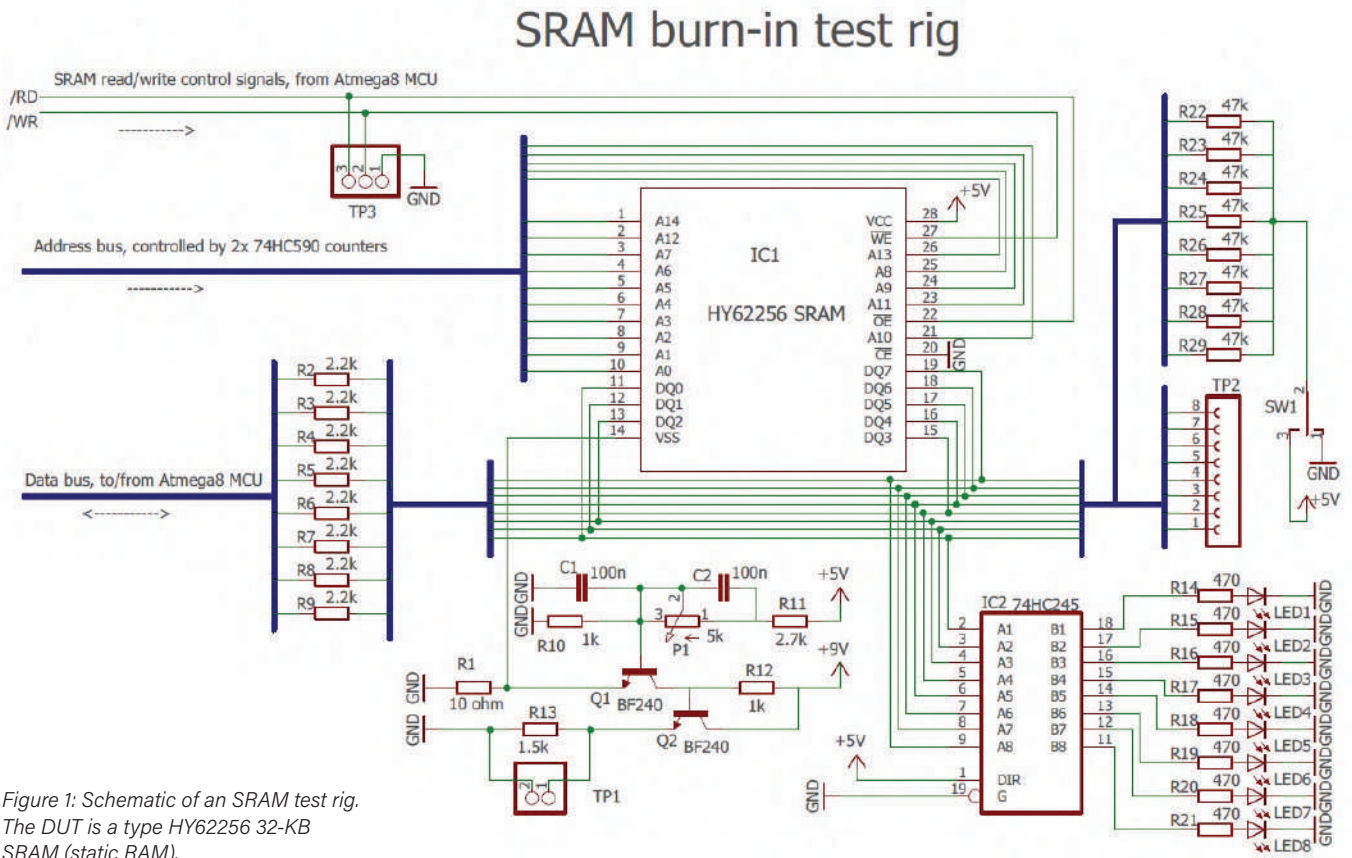


Figure 1: Schematic of an SRAM test rig. The DUT is a type HY62256 32-KB SRAM (static RAM).

SRAM "Burnt-in" Data Recovery

The schematic of a test rig for the experiment is shown in **Figure 1**. The two-stage analog amplifier (first stage Q1 common-base, then Q2 common-collector, a setup for maximum possible bandwidth for fast signals) amplifies the voltage drop on R1, for measurement of SRAM supply current by an oscilloscope hooked up to test point TP1. The whole rig is controlled by one ATmega8 MCU (not shown on the schematics). The address bus is set to a memory address which is to be tested. This is relatively slow, using two 74HC590 8-bit counters. This is okay since the address doesn't need to change quickly. The data bus is read/written by the MCU, which also controls the /RD and /WR commands to the SRAM. These two control lines are used as triggers for oscilloscope measurements, at test point TP3. The resistor array R2...R9 decouples the

data bus, in case it is asserted simultaneously by the MCU and SRAM.

The resistor array R22...R29 is used to pull the address bus up or down (selected on SW1) for measurements of read access and data bus rise/fall times, through test point TP2. Octal buffer IC2 is used to drive the array of 8 LEDs to enable easier monitoring of the data bus.

The key variables to be measured are the power supply current during read/write from/to SRAM, and voltages (actually delay/rise/fall times) on the data bus. Changes in signals measured will indicate the burn-in effects, and hopefully enable the extraction of the bytes written before the SRAM was powered down. The main condition is that the SRAM cells have been holding constant values for a relatively long period. These times can vary depending on the different SRAM ICs used.



Figure 2: I_{dd} current captured on a vintage Tek 466 'scope while writing bytes 0xDE (left) and 0x01 (right) to SRAM (at 200 ns/div).

One 100 Ω , 5 W resistor is attached to SRAM IC for chip heating, with a temperature sensor in between, so the SRAM can be tested at elevated temperatures up to 80-90 °C. You can see it on the test rig photograph in Figure 5.

Changes in rise/fall times of 1-2 ns need to be reliably measured, so a 100-MHz oscilloscope is required. I used a vintage Tektronix

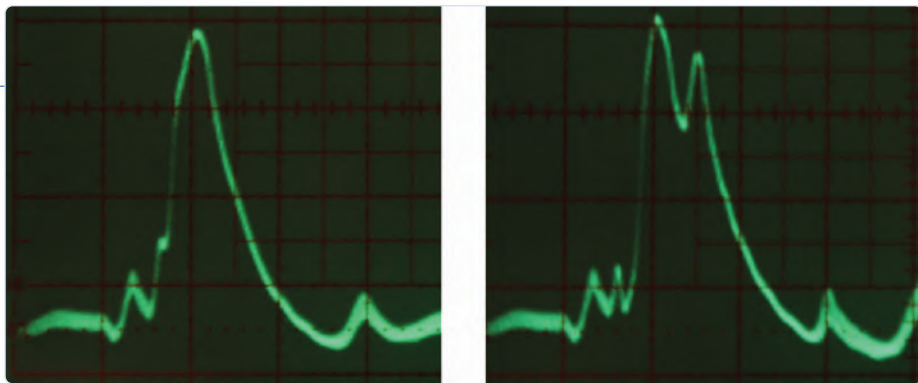


Figure 3: I_{dd} current while reading 0xDE (left) and 0x01 (right) from SRAM (at 200 ns/div).

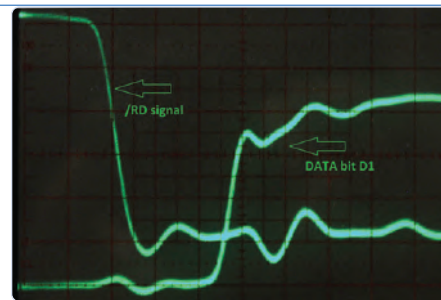


Figure 4: Waveforms of the /RD line and one data bit when reading from SRAM (at 10 ns/div).

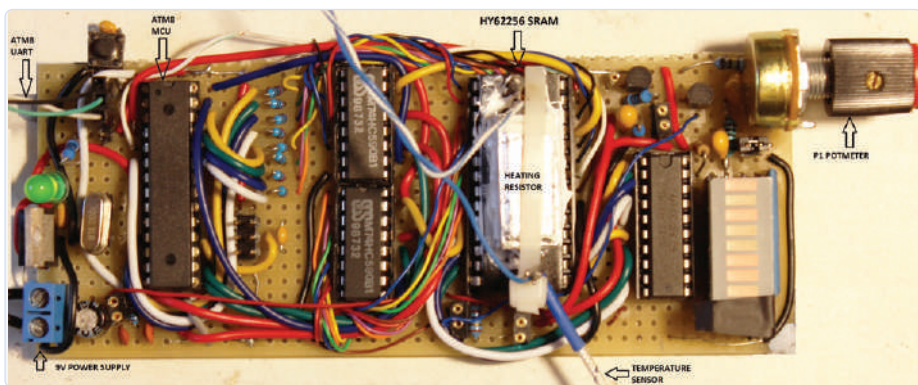


Figure 5: The author's SRAM burn-in test rig.

466 oscilloscope sporting an analog screen-storage option (variable-persistence screen with “flood guns”) for fast non-repetitive signals. There was a nice article in *Elektor Magazine* about this type of oscilloscope in the brilliant *Retronics* section [1]. Storage is not necessarily required though since the read/write test sequences can be made to run in repetitive loops controlled by the MCU. I bought my Tek 466 'scope in Ljubljana (Slovenia) for 150 euros at an electronics flea market.

We start off with some basic oscilloscope measurements on RAM current consumption and the /RD voltage, see **Figures 2, 3, and 4**. Measurements of power supply current when writing from and/or reading to SRAM to cells with burnt-in data (i.e. those which have held the same byte for a long time) or without it (those who have had all of the bits in both states for an equally long time), and comparing the results, can be used to recover the data. Comparison of read-access times and waveforms (Figure 4) between bits on the data bus can also help to recover data.

A picture of my test rig for the SRAM data recovery experiment is shown in **Figure 5**.

I wrote a constant string to the SRAM device and left it to leave all the burn-in effects. I programmed other SRAM cells as counters, incremented every 10 ms. Then I heated the SRAM to 80 °C to augment the burn-in effects by increasing the percentage of hot carriers and left the chip powered up for 12 hours while the MCU kept incrementing counters to avoid burn-in effects on those cells. Twelve hours later, I switched the heater resistor off and allowed the SRAM to cool down to room temperature. The following are the results of the tests carried out on two SRAM cells:

- at address 0x204F, there was a constant value 0x66, which we may expect to have left traces of “burn-in”.
- 0x7FF1 was a constantly incrementing counter, without any trace of burn-in.

Measurements of read-access times (like in Figure 4) showed very small differences between “0” and “1” bits, i.e. less than 1 ns,

so I didn't consider them relevant. Only the MOSFETs used for bistables (i.e. holding a “0” or “1” bit) are affected by burn-in effects, not the other MOSFETs used to access the SRAM cell and transfer it to the data bus when reading. However, measurements of I_{dd} -power supply current while writing different bytes to burnt-in cells gave better results. This method is better for other setups (e.g. for reading SRAM from an MCU) because it doesn't require physical access to the 8 bits of the SRAM data bus.

Once again, the cell with address 0x204F contained the byte 0x66 and experienced a burn-in at 80 °C. The other cell, at address 0x7FF1, had no burn-in (all bits being regularly and timely flipped). The results of the measurements, **Figures 6 and 7**, show that writing a byte pattern with more zeroes to a burnt-in memory cell takes more power than writing the same pattern to a cell without a burn-in. The difference in power required when writing 0x66 (a burnt-in value) and 0x99 (1's-complement of a burnt-in value) to a burnt-in cell is much higher, compared to the other cell with no burn-in. These measurements alone are not enough to establish the exact byte pattern that was burnt into the cell. They require mathematical post-processing (filtering, correlating to known patterns, etc.). The 62256 SRAM is an old, reliable device without much propensity to burn-in effects, nowhere near new highly-integrated devices.

Many more experiments with other types of SRAM, under different operating conditions, are required. Some authors have reported contradictory results, which subsequently indicate that residual burn-in effects aren't well researched and understood. This opens another new and wide field for further research!

Drop the Supply Voltage

Besides these above-mentioned methods, there is one more good approach to extracting the burnt-in data that I haven't tried yet. This is gradually lowering supply voltage down to the point where you get a wrong reading from a certain memory cell, or when writing a byte to it, it fails. Burnt-in cells and cells without burn-in are constantly compared. Usually, only certain bits in a burnt-in memory cell fail to be read/written correctly (e.g. when writing 0x00 or 0xFF), depending on their burnt-in state (0 or 1). This happens because MOSFET gate threshold voltages of burnt-in cells are always slightly increased or decreased compared to cells without burn-in.

Application

The "burn-in" method can be worked out into a secret communication channel enhanced with advanced steganography using newer types of highly integrated SRAM more susceptible to burn-in effects than the "old" 62256 tested here. An SRAM chip chosen for this setup will be okay if it can achieve significant burn-in retention after a few hours of heating at 80 °C (while powered on, with a secret, encrypted message stored) if it can retain the burn-in effects for 10-15 days when powered off (actually removed from the circuit) at room temperature. The communication method would work like this:

1. "Sender" Alice encrypts the secret message for "Receiver" Bob, then stores it on SRAM.
2. Alice heats the SRAM (while powered on, holding the secret encrypted message) to 80°C for 6-12 hours.
3. Alice lets the SRAM cool down to room temperature, while powered on.
4. Alice removes the SRAM from the circuit when it has cooled down.
5. Alice puts the SRAM chip in an envelope and mails it to Bob.
6. If "Spy" Eve intercepts the mail, it just looks like some electronic components package from RS. Even if she tries to recover the burnt-in data, it still won't look suspicious because the message is encrypted. It doesn't mean Alice burned that data in on purpose.

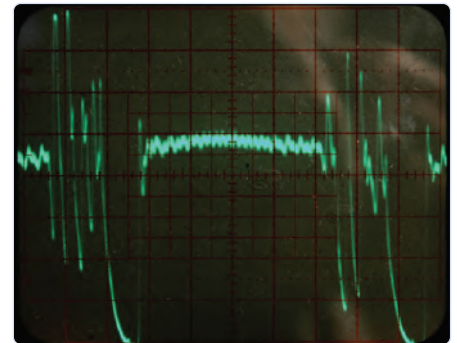
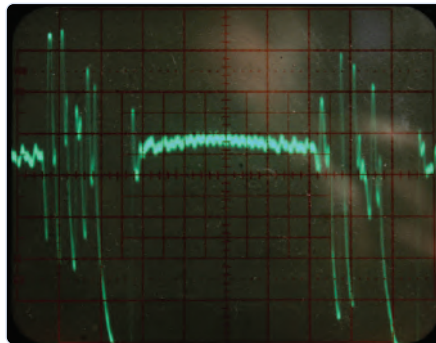


Figure 6: I_{dd} while writing bytes 0x00, then 0xFF to 0x7FF1 (left) and 0x204F (right).

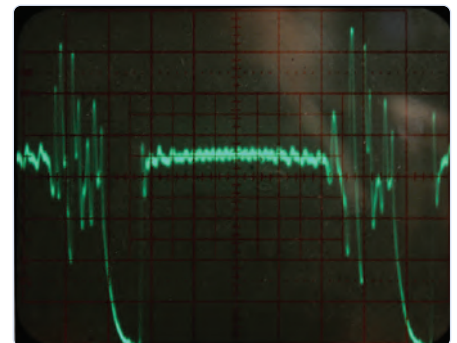
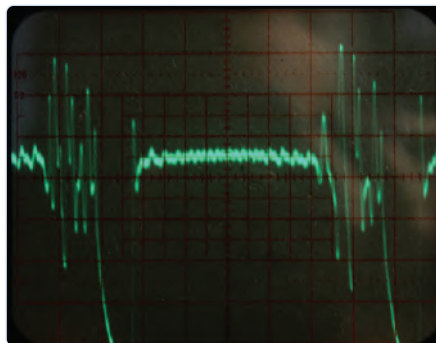


Figure 7: I_{dd} while writing bytes 0x66, then 0x99 to 0x7FF1 (left) and 0x204F (right).

Maybe the chip was just removed from a server where it had been holding some constant data for a very long period – like a few months at 30°C.

7. Bob receives the mail (within a few days) and performs the data recovery procedure as described in this section.
8. Bob decrypts the message.

Can you think of any other ways of how "Sender" Alice, "Receiver" Bob, "Spy" Eve, and others could exploit the SRAM memory retention effects? There are many!

Cold-Boot Attack Demo

The second demo is much simpler than the previous one. The type of attack to follow is likely to work against many old and new types of RAM.

Cooling a chip down as low as -50 °C will not significantly reduce the conductivity of moderately-doped silicon microcircuits, so the chip will continue to operate normally,

but will retain the data after disconnecting the power because the discharge of MOSFET gate capacitances will be much slower. The same rig will be used for the experiment, only without the heating resistor. The eight LEDs on the bar graph will display various running-light patterns directly out of SRAM. If they continue to run in regular patterns (after a power cycle) it means the SRAM contents got preserved. If irregular, erratic patterns appear, it means the SRAM contents were lost. I measured the maximum data retention time at around -30°C, which is easy to reach using the well-known "KÄLTE 75" freezing spray (Figure 8).


At room temperature, there is only 0.1 s data retention time, but up to 10 s can be expected at -30 °C.

The SRAM IC is first cooled down to around -30 °C and the power-cycle time is gradually increased while maintaining



Figure 8: Sub-zero cooling spray used in the cold-boot attack demo.

equipment inside your home lab (like cold-boot attacks), while some other attacks including SRAM burnt-in data recovery may need more sophisticated hardware and mathematical post-processing (although within the low-budget spy's reach!) to be fully successful in real life, not just as proof-of-concept demos.

that good encryption can be completed on a low budget nowadays. As the technology advances and becomes cheaper and more accessible, the process works much in favor of Alice and Bob. 

210628-01

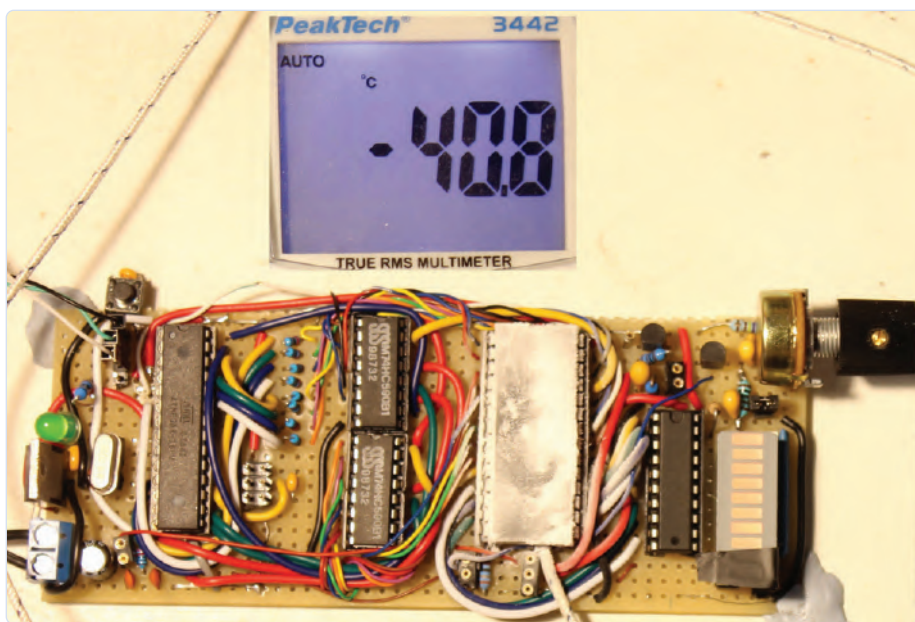


Figure 9: The SRAM IC was "deep-frozen" to achieve the 10-seconds data retention time.

the temperature with the spray. The 10-seconds retention time was reached at -40 °C (Figure 9). These results were to be expected since the power supply pins remain "shorted" through some kΩ-range resistance while powered down, so the V_{dd} pin may be considered as "grounded".

Surprising Simplicity

As you can see, some practical attacks can be fully accomplished with very cheap

Please keep in mind that new highly-integrated electronic systems are more susceptible to any of these attacks demonstrated (except maybe for modern laser and ink-jet printers compared to old dot-matrix or daisy-wheel). Improving on the ideas shown here may lead you to design your unique attack devices and procedures. There is plenty of work to do and no more excuses for 21st-century design engineers' typical apathy and depression! I am sure

Questions or Comments?

Do you have any technical questions or comments related to this article? Email the author at luka.matic@gmail.com or Elektor at editor@elektor.com.

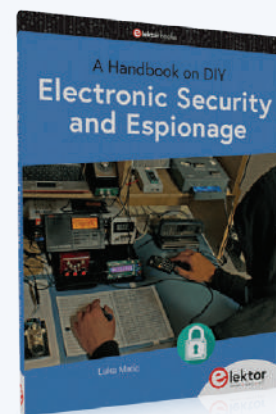
Contributors

Text: **Luka Matic**
Editor: **Jan Buiting**
Layout: **Giel Dols**



RELATED PRODUCTS

- > **Book:** L. Matic, *A Handbook on DIY Electronic Security and Espionage* (SKU 19903) www.elektor.com/19903
- > **E-book:** L. Matic, *A Handbook on DIY Electronic Security and Espionage* (SKU 19904) www.elektor.com/19904



- > **E-book:** J. Buiting, *Retronics, 80 tales of electronics bygones* (SKU 16885) www.elektor.com/16885

WEB LINK

[1] Retronics: The Tektronix 564 Storage Oscilloscope (1963), Elektor 05/2011: www.elektormagazine.com/magazine/elektor-201105/19598

Component Identification

Tips & Tricks, Best Practices
and Other Useful Information

By David Ashton (Australia)

Being able to identify the parts on a circuit board is primordial for repairing it. Developing your identification skills also enables you to create a stock of spare parts that might come in handy someday. However, it is not always easy to determine what is what. Here are some tips to help you out.

Many readers will, like me, be hobbyists who scrounge parts off old electronic equipment. I got interested in electronics in my early teens — about 50 years ago — and my father, who was in sales at an accounting machine firm, got me scrap boards from the techs there. The boards contained transistors, resistors, diodes and a few capacitors. They had the leads bent over on the underside of the board and even the large soldering gun my dad also got for me had problems removing some components. However, this was in Zimbabwe (then Rhodesia), which was not the centre of the electronics universe. Components were expensive and not always easy to get, so these boards were gold for me.

The transistors were in a very weird package, but with a simple transistor tester I built from an article in a magazine, I identified them as NPN types. However, they were marked “B686” and had a dot of paint — brown, red, orange, yellow or green — in a dent on top of the case. Now, in those days, the Internet didn't exist, and data books were rare and expensive. I did know that transistors marked “Bxxx” were often from the Japanese 2SB series, but when I referred to my copy of the venerable *Towers International Transistor Selector* book,

the 2SB686 was a PNP power transistor and mine were NPN signal types. So, identifying the actual type number of these transistors was a challenge. And not an easy one!

Success came when by chance I saw a table of transistor data in the *Practical Electronics* magazine I got at the time. There was the 2N2926, with the same case, with a note that the dot of paint on top of the case showed the gain range. The 2N2926 (**Figure 1**) is a strange-looking transistor and it had to be the same, even though mine were not marked as such. I confirmed that the gains of my transistors corresponded to those shown in the table, and that sealed the deal. The 2N2926 with its dot of paint is such an unusual transistor that it had to be that, even with a different marking.

I've worked in electronics and telecoms most of my life, and still strip old boards for parts — professional equipment often yields high quality components. Since my (misspent?) youth, I've had a lot of challenges to identify even weirder scrounged components, not always successful. But I am passing on here a few of the techniques I have depended on over the years. And I have included a component identification quiz for you to test your own skills. This has a real mix of component types from very old ones to newer SMD types, and I've included examples of the problems and techniques I describe here.

Collect Component Data

When I started, data books were difficult to find, and worth their weight in gold. Not as important now with the Internet at hand, but if you see any useful information — colour codes, manufacturers' information, datasheets, etc. — keep it filed away in your favourites or as a printed copy. I still have the table of transistor data from all those years ago.

These days, get to know your way round the Internet. There are plenty of good datasheet sites out there. Google is your friend! But help Google to help you. If you're looking for a datasheet, put “datasheet” in the search box. And use the minimum part number you can. I had

developer's zone

Tips & Tricks, Best Practices and
Other Useful Information



some ICs recently labelled "2026-1SM," which, as I had several, I had determined was probably the part number. But I put "2026 datasheet" into the search engine and right away came up with the datasheet for the MIC2026 by Micrel (now part of Microchip) — a dual-channel power distribution switch.

Some datasheet sites take your part number and specify datasheets that are either an exact match, start with or just contain the data you put in — this can be useful to narrow down your search. I save most datasheets to my hard drive — you never know when you're going to have to hunt for them again — but this is a personal preference. I've given a link to my most used datasheet site at the end, and keep a few sites in my favourites, along with some manufacturers' sites.

How to Read Datasheets

Most datasheets start off with a description of the part, then the absolute maximum ratings. Then follows the nitty-gritty of how to use it, pinouts, etc. The package information is usually at the end, and you often need to look at that to make sure that what you've got is indeed what that datasheet is about — the same number of pins and the same package.

Manufacturers don't all make the same devices in the same package, so you may have to hunt around. Usually, but not always, a part made by one manufacturer will have the same specs as the same part made by another manufacturer, so if you can, find a datasheet from the manufacturer of your component.

Know Your Components

Some components look a lot like something else. These days I can look at a component that looks like a resistor and say "That's a capacitor" or "That's an inductor" with a fair degree of accuracy, based on the shape and the colour of the component.

Most people know that something marked "2Nxxxx" is a transistor, and an IC-looking thing with four or six pins is probably an opto-isolator. And I know that a transistor-looking component marked xxNyy (e.g., "35N60") or one marked IRFxxx (e.g., "IRF540") is a FET. This comes with experience.

Just while researching this article, I came to the realization that a "V" on an IC often means it's made by Vishay, which I'll remember as it may save me some time in future.

How to Test Components

About the time I encountered those 2N2926s, I built myself a simple transistor tester with an old meter, a switch, a 500 kΩ pot and a 1.5 V battery. I still have it today, but nowadays most digital multimeters have a transistor tester built in, and some can test capacitors as well. FETs are a little more difficult but are possible to test in the average home lab. I have a basic LCR meter, which I use a lot, though I'd like a better one.

If you can identify a component as a transistor, FET, capacitor, etc. you are halfway there, and you may be able to use it even without further identification. Some SMT (Surface Mount Technology) components — capacitors in particular — have no markings at all so you need to measure them to be able to use them. A tweezer probe for your meter can make it a lot easier to test SMD parts, and the Elektor Store has a very tasty digital tweezer tester that measures just about anything. There is a link to it in the **Related products** textbox.

ICs, of course, need specialized testers, but you can make yourself an op-amp tester and you can buy logic IC testers if you get and use lots of logic ICs. A part number and datasheet are first prize for ICs and transistors, but lots of components, particularly passive ones, are usable if you just know their value. Testing components is a subject in itself.

Make the Best Use of Component Print

There are usually a few numbers on a transistor or IC and it's worth a bit of effort to identify which is the type number. Prefixes are often omitted on small SMDs. Get to know manufacturers' logos. Going



Figure 1: 2N2926 transistors. The colour specifies the transistor gain H_{fe} : Brown 35-70; Red 55-110; Orange 90-180; Yellow 150-300; Green 235-470.

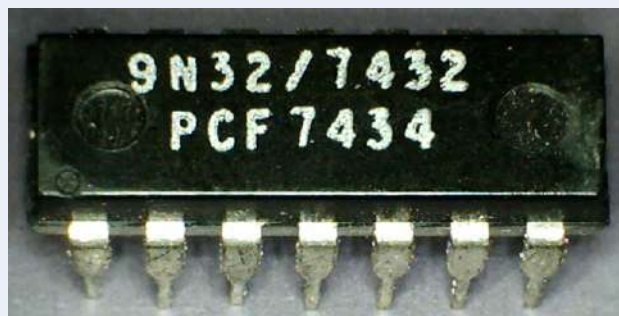
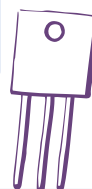
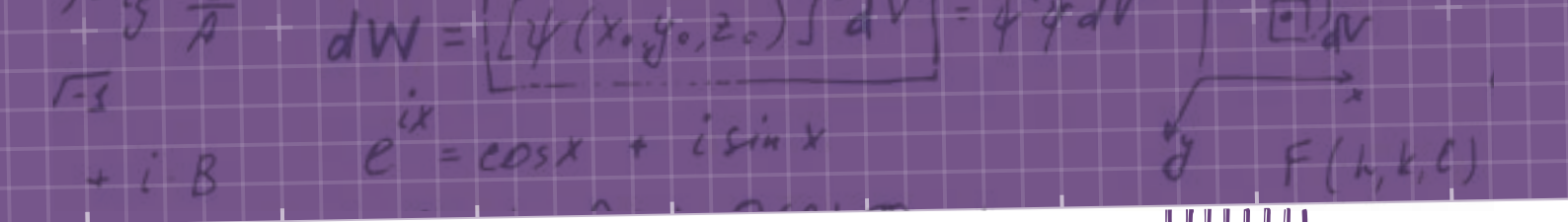


Figure 2: Is this a 7432 or a 7434? You'd be forgiven for thinking that the PCF is a manufacturer's prefix, but it's not. (It wasn't in those days, anyway!) There is no TTL 7434. It's a 7432 made in the 34th week of 1974. The 9N32 is another clue, many 74xx ICs were also labelled 9Nxx, though it's very difficult to find any information on this.





straight to the manufacturer's website can save you lots of time. Most components have a date code which used to be year and week (like 8634) but these days they can be cryptic batch codes. (In the old days, a TTL 74-series logic IC made in 1974, with a 74xx date code (**Figure 2**) could be a puzzler!)

If you have a few components of one type, look for a code that's the same on all of them — that will be the part number, the other codes will be date — or batch codes of no interest.

Values on passive components are either shown directly (like 47 pF) or as figures or resistor colour code in the form *Digit1, Digit2, Multiplier* (being the number of zeros) on the component. So SMD inductors often have the value marked in microhenries in this fashion, so **3R3** is 3.3 μ H and **333** is 33 mH (33,000 μ H). Capacitors may be marked in picofarads. A tantalum capacitor labelled **227** is 22×10^7 pF = 220 μ F. Some components may have five or six rings of colour code, but the Internet is a great help in decoding these.

Most small SMD capacitors are not marked at all, so use your component test skills and equipment to verify them. And get a magnifying glass or a USB microscope (which is what I used to take most of the photos for this article) – it makes it much easier to see the tiny writing on small components.

The internet has many resources to assist you – search for “IC Manufacturers logos” or “SMD codes” if you need more information. And look up “EIA-96” to decode SMD resistors with what looks like a weird 2-number plus 1-letter code.

Consider the Context

If you're stripping parts off boards, or otherwise know where the component came from, that may give you a clue as to what the part is. A power supply is likely to have a switched-mode PWM IC, whereas an audio board is more likely to have op-amps.

Don't Expect to Identify Everything!

I have a bag of transistors labelled **0V8F** which have stubbornly refused to be identified. SMD components can be difficult or impossible to identify, as they often have shortened part numbers on them. Even with the considerable resources on the internet, they are not easy.

Be Selective

I mentioned the boards I got as a kid, with bent-over component leads. I religiously unsoldered all of them. These days I won't touch such components unless they are really special; it's not worth the effort.

Electrolytic capacitors should always be tested, especially large power supply types, and look for domes on the tops — a dead giveaway that they have gone dry or leaky.

Older components such as carbon resistors are really not worth keeping, and older electrolytic capacitors will often be much larger, for their ratings, than modern types.

Many modern SMD ICs have very fine pitch leads or are ball-grid array (BGA) types which need specialized equipment to install and remove on boards. So if it's not something you can see a use for, junk it.

If you think it might be something you can use, identify it to be sure before you go to the trouble of unsoldering it.

Being able to identify and use components from old boards can well repay the time you spend doing it. You can get high-quality components and if you store them systematically, you can often avoid having to buy parts for a project. In addition, you can often use these skills when attempting repairs on circuit boards.

210024-01

Contributors

Idea, text & illustrations: **David Ashton**

Editor: **Clemens Valens**

Layout: **Giel Dols, Harmen Heida**

Questions or Comments?

Do you have technical questions or comments about his article?

Email the author at stn564@yahoo.com.au or contact Elektor at editor@elektor.com.



RELATED PRODUCTS

- **Andonstar AD407 HDMI Digital Microscope (SKU 19079)**
www.elektor.com/19079
- **Miniware DT71 Mini Digital Tweezers (SKU 19422)**
www.elektor.com/19422
- **OWON OW16B Digital Multimeter with Bluetooth (SKU 18780)**
www.elektor.com/18780

WEBLINKS

- [1] [Good datasheet site with lots of options:](http://Good datasheet site with lots of options: www.alldatasheet.com/)
www.alldatasheet.com/



To the Quiz →

TAKE THE COMPONENT IDENTIFICATION QUIZ

Scales are in millimetres. See how you do!



A - Can you tell what this is just by looking at it?



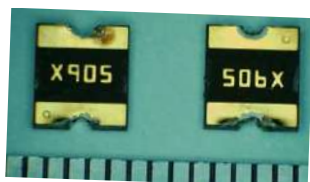
B - And this one?



C - What is this and what's its value?



D - What is this strange-looking IC? Weird package, but nice part number (or is 431 the part number)?



E - 506X or X905? These SMD parts show very low resistance (a few ohms). What are they?



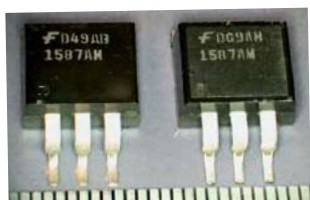
F - What would this IC be?



G - This is a weird one. There are the figures 431 on it which are hardly visible.



H - Are these the same type of component, or different?



I - What are these? Can you find a datasheet?



J - Is this a tantalum capacitor? Or something else?



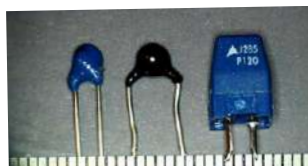
K - Marked D256. Hint: the angled view shows the thickness of the IC.



L - This one's pretty easy. But a hint: who's the manufacturer?



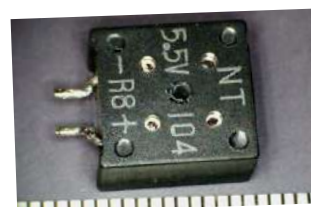
M - Remember I told you about this one? Any ideas?



N - These are similar components but what are they? Tantalum capacitors?



O - No markings. But what's this for? What are the "handles"? Heat sinks?



P - This case has 4 holes in it (between the lines of writing). Is it a buzzer, or a temperature or humidity sensor?

QUIZ ANSWERS



K – The thickness of the IC is a clue. ICs like this are usually optoisolators. Another clue is the “V” usually indicates an IC made by Vishay – See D above. This can sometimes save you a fair bit of time. Its full name is ILD256T and it is a dual channel, AC input optoisolator with well-matched Current Transfer Ratios (CTR). Nice.

L – See the Texas Instruments logo just before the 49 on the first line? Go to the TI website, put in 2064, and you get three possibilities: a power board, a power distribution switch with eight pins and this – the TLE2064 – an improved version of the TL064 quad JFET input op-amp. As usual, no prefixes on SMD components.

M – Here’s my unidentifiable 0V8F transistor – in standard TO-92 case – referred to in the text. Ten points to the first reader who can identify it with manufacturer and datasheet. I have a few of these and the type number is 0V8F, the other numbers are a date code and vary. Yes, I’ve tried 0V8F!

N – The small blue and black ones are NTC thermistors. As the temperature goes up the resistance goes down. They’re from air-conditioning sensors. The large one is a PTC thermistor – more used for protection. If excess current flows, they heat up and go high resistance to limit the current – like the polyfuses in E above.

O – I couldn’t resist including this one. Did you think it was a current sensor? You’re right! The pic is from the Allegro ACS756 Hall Effect Current Sensor datasheet. The “handles” are the current path, and the pins are the Hall sensor. Up to 3 kV isolation. Nice for power supplies.



P – This is a supercapacitor. 104 = 100,000 μF = 0.1 F. Note the value is in μF , not pF as you see on other caps. 5.5 V is a common voltage for these. They’re usually used for memory or real-time clock power backup. Here it is shown with a more usual form factor 0.1 F supercap. Why the holes in the case? I don’t know either!

handy self-resetting fuse device. The codes on them were no help.

F – With experience, anything that says 431 is usually the TL431 voltage reference from Texas Instruments, or a second-sourced one. A widely used, versatile, voltage reference that can also be used for voltage sensing in power supplies. This is the SMD SOIC-8 version – usually they are in the TO-92 transistor case.

G – This one puzzled me for a long time. Just visible is the number 431 on the red blob. It turned out to be a 430 V VDR (Voltage Dependent Resistor), a type of surge arrester. So it meters open circuit, and very little capacitance. Physically very small so I wouldn’t expect this to absorb a high current surge.

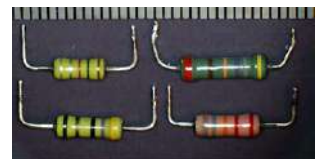
H – They are almost identical in size and colour. On the left is a capacitor, on the right is a VDR (Voltage Dependent Resistor). The cap does have a value (102 = 1 nF) printed on it, and it’s rated at 2 kV. The VDR has 241 (24 and one zero) – so a 240 V VDR. You’d be unlikely to see a 1000 V VDR, and it wouldn’t have another voltage on it! Again, experience helps here, and never assume!

I – The 1587AM is common to both parts. Search for “1587 datasheet” and you are directed to Alldatasheet.com, select parts that end in 1587 (*1587) and it comes up with the LMS1587 LDO voltage regulator. That’s by Texas, but if you recognise the Fairchild logo and google “Fairchild 1587” you get directed to the Fairchild RCI1587, which is what this is, the same component.



J – This is a tantalum capacitor – 20 μF – from the old days when they used to be colour-coded. Here are some of his mates, 2.2 μF , 3.3 μF and 5 μF . Only two in preferred values, and very weird colour codes! Be wary of old “tantis” like this, they tend to go short-circuit. But they’re prettier than the newer ones!

A – This is a tube ceramic capacitor – 22 nF. Not quite resistor shape and the wrong colour. Shown here with – clockwise – 47 pF, 1 nF, and... a 486 k Ω 1% resistor, which is the most un-resistor looking of the lot – much longer than a normal resistor! It fooled me when I first saw it. So never trust yourself, always test!



B – As the capacitor in A was too long for a resistor, this is too short and fat, and the green-blue colour would be unusual for a resistor. It’s an inductor of 680 nH. Here it is (top left) with (clockwise) 470 nH, 47 nH and 820 nH. The 47 nH is the usual cream resistor colour, but a bit too short for a resistor.

C – This is a very old resistor, using the body-tip-spot colour code. Red Black Yellow = 200 k Ω and no tolerance, so probably 20%. It actually measures 225 k Ω – only 12.5% out! Probably not worth keeping except for sentimental value... What do you mean, preferred values? It’s probably 80+ years old!

D – These needed a lot of work. You can easily find the LH1540 – an opto-isolator, in various packages but not this one. It also has 431 on it – was it maybe a weird version of the TL431? Eventually at the bottom of a Vishay datasheet I saw a picture of that package (called an 8 PowersOIC) with no other information except a link to the LH1540ACD datasheet. That in turn led me to the LH1540ACD datasheet which refers to this particular package. Sometimes you have to dig.

E – Low resistance, huh? On a whim I put it on my power supply and slowly increased the current. At about 700 mA, it gets warm and goes fairly high resistance. It’s a Polyfuse – a very

DIY Touchless Light Switch



By **Mathias Claußen** (Elektor)

It's an idea born out of a pandemic: a light switch that works by interpreting hand gestures. No physical contact means fewer opportunities for viruses and bacteria to be transferred from person to person. This is important for busy areas in large workspaces. This article outlines the concept and the practical implementation with an ESP32 and a shield that evaluates hand gestures.

When the first wave of COVID-19 struck in early 2020, measures were taken in offices to contain the spread. Surface cleaning and disinfection has become a daily ritual. While it is easy to wipe down coffee machines and kettles before and after use, it is more difficult to ensure other surfaces such as light switches

in open-plan offices and corridors could not become pathways for the virus to proliferate.

Thinking about it in the Elektor Lab, I realized that I could solve the problem by installing switches that required no physical contact.

Warning

All circuits presented in this article operate on mains voltage. Neither the circuits nor the circuit boards shown here have been tested with regard to their function or their safety. It is not advisable to build or operate the circuit boards without first having them checked. Should you decide to build a project based on this design, it is important to be aware of the dangers. Work on this project entirely at your own risk. It should be attempted only by qualified engineers or designers!

To do so, I found a 3D Gesture & Tracking HAT from Seeed Studio for the Raspberry Pi (**Figure 1**) in the Elektor Store [1].

The MGC3130

A contact-less light switch called the The Open Smart Switch [2] by James Rowley and Mark Omo was posted on the hackster.io platform at the beginning of 2019. The Microchip Technology MGC3130 was used in the project. The 3D Gesture & Tracking HAT for the Raspberry Pi that we plan to use also uses the same chip. The chip and its characteristics have been discussed in detail on the ElektorTV channel [3].



The MGC3130 is an E-field sensor. The chip sends out a signal from an electrode which produces an electric field on the surface of the gesture/touch pad. Five electrode areas positioned around the pad are used to detect the field and sense any subtle changes when objects such as a hand come close to the pad and disturb the field. **Figure 2** shows the principle function of the sensor and the generated field. In addition to recognizing and calculating the position of a finger, 3D gestures can also be evaluated such as an 'air wheel' and a 'flick' in all four directions. A flick consists of moving a finger in the air across the sensor.

The chip itself can be calibrated and configured using tools provided by Microchip. All the information you will need to use the chip for your own design is available on the product page [4].

Its communication with the outside world is via an I²C interface. In addition to the data and the clock line, the chip also needs a reset and a bidirectional interrupt line. Details of the interface are provided in document DS40001718E [5] from Microchip. You can design your own custom pad for use with the chip but note that it is a multilayer board made up of least four layers. In this case it will then be necessary to parameterize and configure the chip to match the new pad design.

The MCU

When it comes to choosing the type of microcontroller, we really are spoiled for choice these days. The range extends from the Raspberry Pi Pico, an STM32 Blue Pill, the proven ATmega328, to the ESP8266 or the ESP32 (to name but a few). The application does not require any serious number crunching and only outputs a single signal to switch the relay.

We also need to consider how the switch will function; will it operate independently to switch a single lamp or be integrated into an existing home automation setup? The second option, integrated into a control system such as the ESPHome would be advantageous



Figure 1: The 3D Gesture & Tracking Shield.

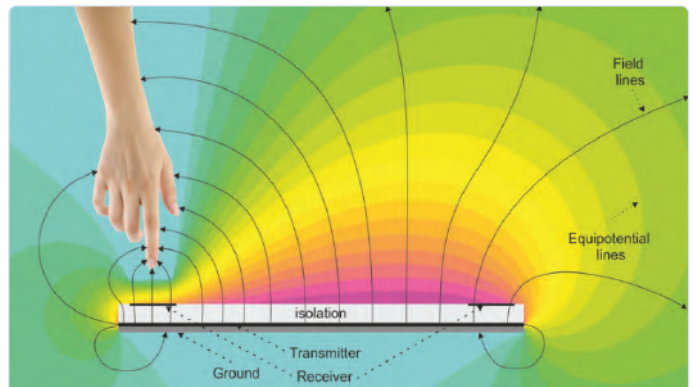


Figure 2: The MGC3130 Sensor field (Source: Microchip).

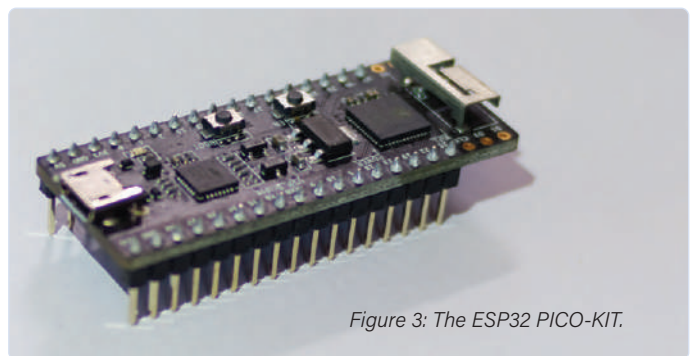


Figure 3: The ESP32 PICO-KIT.

because the lamp or load will then be switched from a central point. The question also arises as to how any firmware updates can be made to the system. With the ESP8266 or the ESP32, this can be made OTA via a Wi-Fi link.

An ESP32 variant has been chosen for this project; it has Wi-Fi, Bluetooth, enough on board Flash memory and can perform updates via Wi-Fi; the ESP32 PICO-KIT is shown in **Figure 3**.



Figure 4: Flush mounting back box (Source: Würth [9]).

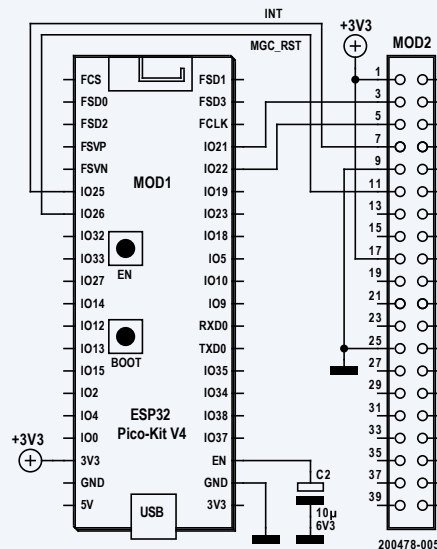


Figure 5: The prototype schematic.



Figure 6: Plugging it all together.

The Basic Concept

As with any project, it's necessary to set out the basic concept and check out all available components that could be used to accomplish the task. Here we already have a sensor in the form of the 3D Gesture & Tracking Shield for the Raspberry Pi and an ESP32, in the form of an ESP32 PICO-KIT. A relay is also required to switch the light on and off and lastly we need a power supply. All we need now is some firmware and the jobs done!

If only it were that simple. Oftentimes during the development process we come up against unforeseen problems that need to be overcome. The development process can be broadly divided into three disciplines that work together to build the finished product. Firstly there is the software which evaluates sensor data and makes appropriate actions. Next is the electronic hardware and PCB on which all the components are mounted. A circuit diagram is produced showing how the components are connected. Ideally, feedback from the software developer has already been sought

regarding pin assignments and any special requirements. It is often the case that mistakes in the hardware layout can be corrected by making small changes to the software but this can be a source of annoyance to the software developer.

Lastly, there is the mechanical design of the completed assembly and enclosure into which everything must fit. Often the size of the enclosure is not critical and we can select or fabricate one that will be big enough. For some designs, however, we may need to fit the design into a standard enclosure such as an ordinary electrical back box where it can then be integrated into the domestic wiring installation. In this project, our initial plan was to fit the complete design into a volume measuring 65 mm x 55 mm x 25 mm. This however does not take into account some of the common flush-mounted boxes (Figure 4) used in some European countries. To give the design more universal appeal, it would be beneficial if the assembly could be made small enough to mount inside this type of enclosure also.

Breadboarding the Initial Design

The circuit diagram from Figure 5 shows the initial hook up of the 3D Gesture & Tracking Shield for the Raspberry Pi to the ESP32. It is set up on a breadboard (Figure 6) to test its functionality. There are several libraries available to use with the MGC3130. The library used in this project comes from Sreed Studio [6] and is intended for use with a Raspberry Pi.

For the ESP32, a few adjustments need to be made to the I²C interface and the I/O pin assignments. After these modifications, the library could then be used for initial tests. Figure 7 shows the positional coordinates of movements for a recognized 'flick gesture'.

The PSU and Relay

During breadboarding and testing the prototype, it wasn't necessary to worry too much about the power supply and the relay that would be included in the final design. The finished controller will operate from mains voltage. To do this, we need to include a power supply that provides 5 V or 3.3 V with enough power for the ESP32, the 3D Gesture & Tracking Shield and relay.

```

26  mgx3130_rst_pin = resetPin;
27  xTaskCreate( Touchinput_Task, "MGC3130drv", 4096, NULL, tsKIDLE_PRIORITY, &
28  configASSERT( xHandle );
29  }

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
x: 48548, y: 65534, z: 38456
x: 47465, y: 65534, z: 34252
x: 46156, y: 65534, z: 33950
x: 44656, y: 65534, z: 33498
x: 43026, y: 65534, z: 32857
x: 41337, y: 65534, z: 32204
x: 39667, y: 65534, z: 31760
x: 38103, y: 65534, z: 31560
x: 36732, y: 65534, z: 31502
x: 35621, y: 65534, z: 31525
x: 34798, y: 65534, z: 31599
x: 34243, y: 65534, z: 31708
x: 33899, y: 65534, z: 31841
x: 33696, y: 65534, z: 31986
x: 33569, y: 65534, z: 32119
x: 33467, y: 65534, z: 32215
x: 33368, y: 65534, z: 32265
x: 33237, y: 65534, z: 32278
x: 33103, y: 65534, z: 32277
x: 32972, y: 65534, z: 32275
x: 32857, y: 65534, z: 32284
x: 32764, y: 65534, z: 32304
Gesture: FLICK_SOUTH_TO_NORTH, class: FLICK_GESTURE, edge flick: no, in progress: no
x: 32687, y: 65534, z: 32331
x: 32611, y: 65534, z: 32358

```

Figure 7: Data output from the hand gesture pad.



Figure 8:
Hi-Link AC/DC converter.

One of the more compact power supply modules is the HLK-PM03 or HLK-PM01 from Hi-Link (**Figure 8**). These deliver 3 watts of output power, i.e. 600 mA at 5 V or around 900 mA at 3.3 V. The ESP32, requires around 500 mA (2.5 W at 5 V, 1.65 W at 3.3 V). A Panasonic type ADW12 relay will be used, which needs an impulse of around 67 mA at 3.3 V or 40 mA for the 5 V (220 mW) version. The 3D Gesture & Tracking Shield itself requires another 20 mA at 3.3 V (66 mW). If we add everything up, it comes to a minimum of 2.8 W if the ESP32 PICO-KIT is powered at 5 V. This brings us close to the maximum output power from this 5 V power supply.

The ESP32 PICO-KIT consumes less power when run from a 3.3 V supply because the on board LDO regulator is no longer needed and will not be powered. For this reason we will use a 3.3 V PSU such as the HLK-PM03 for the final design. It can supply 900 mA maximum. A Panasonic ADW1203HLW is selected as relay. This single-pole latching relay has a contact rating of 16 A at a maximum of 277 VAC. At 24 mm x 10 mm x 16 mm, this is quite compact. Only a short pulse is required to set or reset the relay and no permanent current is required through the coil of the relay.

The Latest Schematic

The preliminary circuit diagram is shown in **Figure 9**. As yet we have not included any protection around the HLK-PM03 power supply module. Typically a thermal fuse and varistor will be fitted to the AC input side and the output will require a fuse and some voltage smoothing capacitors. If a fault develops in the module without these protective measures on the input side, it could lead

to a fire. Poor output supply voltage regulation and filtering can significantly reduce the Wi-Fi range or trigger a software crash. The finished power supply will include these modifications.

A 10 μ F capacitor is placed between the GND and EN pin on the ESP32 to prevent timing problems occurring in the reset sequence which can prevent the chip going into programming mode. It also helps in a situation when the ESP32 occasionally refuses to be programmed from the USB micro port.

As already mentioned, the ESP32 PICO-KIT board includes an AM1117-3.3 regulator to provide 3.3 V from 5 V for the ESP32. This regulator can withstand 3.3 V on its output pin when it is unpowered. However, in our circuit diagram, a Schottky diode (D1) is connected between the 3.3 V supply and 5 V supply so that any regulator fitted to any other type of microcontroller board could not be damaged by a reverse voltage condition. Its inclusion will draw current from the 3.3 V supply through the regulator.

The 3.3 V relay requires a drive current of at least 67 mA to switch. This is significantly more than the pins of the ESP32 can provide. A transistor driver stage (T1 and T2) is used to power the relay. The

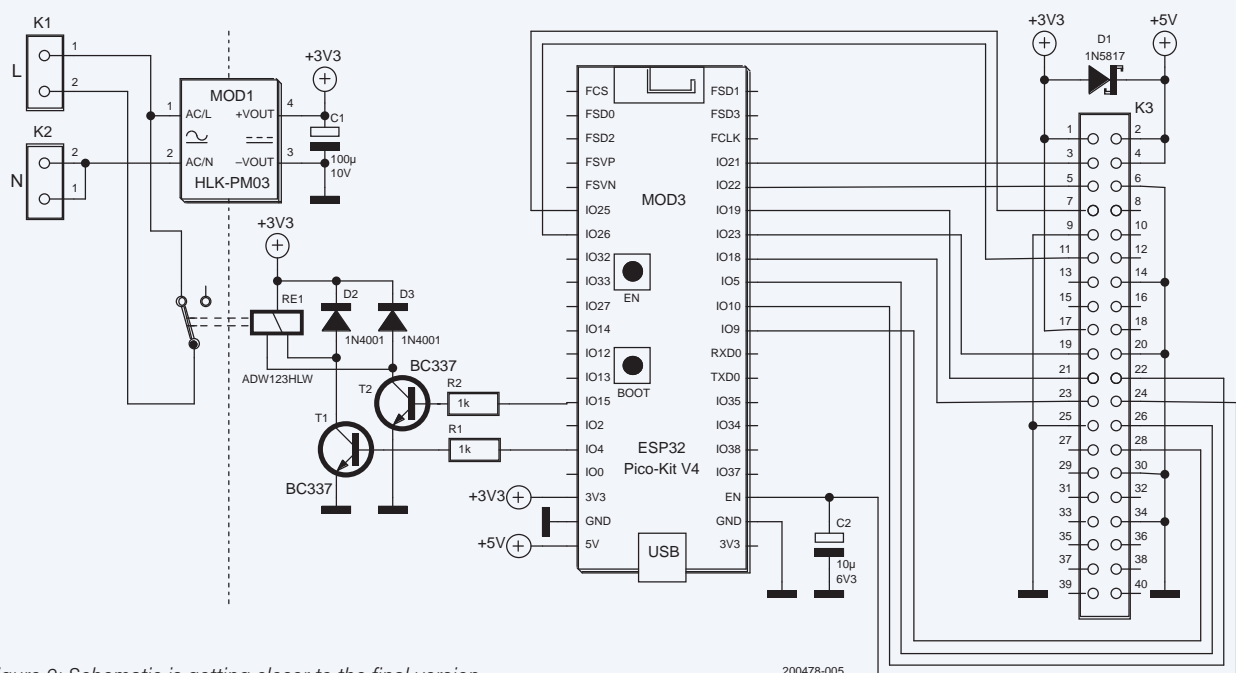


Figure 9: Schematic is getting closer to the final version.



Figure 10: The basic firmware blocks.

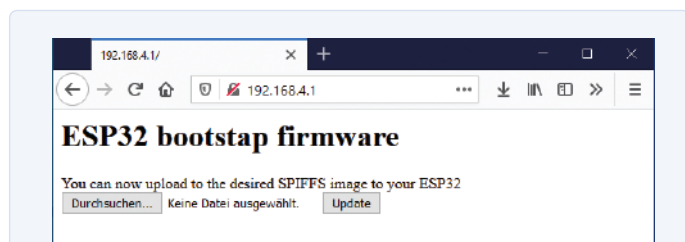


Figure 11: SPIFFS image upload using a Web browser.

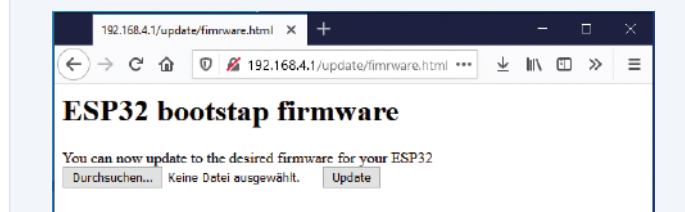


Figure 12: OTA firmware Update using a Web browser.

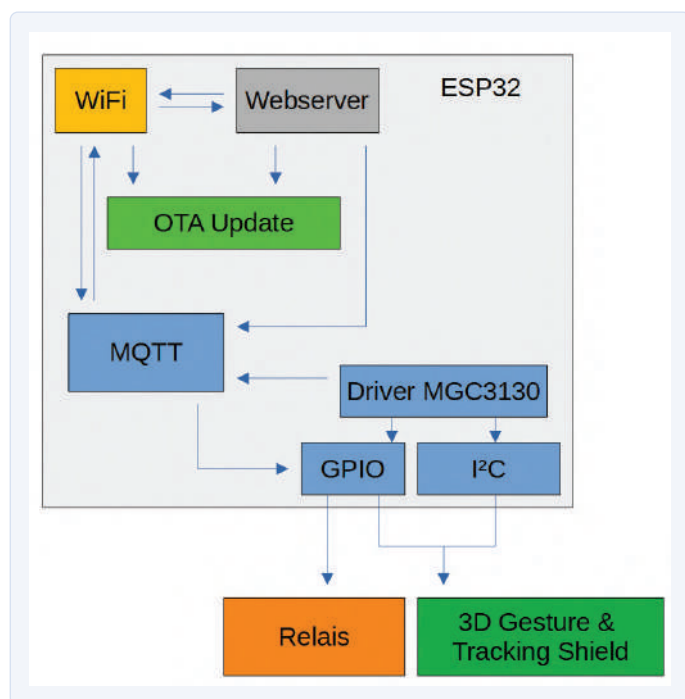


Figure 13: Firmware module linkage.

freewheel diodes D2 and D3 are essential to prevent the transistor being destroyed by reverse voltage spikes when current to the relay coil switches off.

The 3D Gesture & Tracking Shield is connected to the ESP32 via the two-row pin header K3. The 3.3 V, I²C signals and two I/O pins are the only connections required for the shield itself. The shield already includes the necessary pull-up resistors for the I²C bus. The board has been designed to provide SPI signals and the other necessary I/O pins for possible future addition of a 3.5" TFT monitor.

The flexibility of the I/O matrix facility in the ESP32 makes the connection of peripherals very easy. For the 3D Gesture & Tracking Shield, Pin 21 and 22 are used for the I²C bus, Pin 25 as the interrupt and Pin 26 for resetting the MGC3130. If the option to fit a TFT monitor is not required, we would be able to manage with a significantly smaller ESP32 variant or even use an ESP32-C3 module.

The Firmware

After the pins of the ESP32 have been assigned, it is necessary to compile the firmware for use. We can use library routines to create a web server which generates two basic web pages and allows us to carry out OTA updates to the firmware. The modules of the basic firmware can be seen in **Figure 10**. This approach avoids the need to rewrite and debug some of the same basic functions that are used on many similar applications.

The firmware generates two basic web pages. One is used to write a new SPIFFS image to the ESP32 (**Figure 11**) and the other to provide ESP32 firmware update (**Figure 12**).

The firmware as it stands just performs the basic switch functions. On top of this there is another component to evaluate the MGC3130 information and a minimal web page to configure the MQTT messaging and another which allows you to switch the load on or off via a web browser. The interaction of the modules can be seen in **Figure 13**. The light switch can therefore be controlled automatically or manually.

Component Tetris

Now that the firmware and circuit diagram are ready, we need to make sure all the hardware can fit in the space available. Everyone has their own preference for PCB design tools and should use the one they are familiar with. In my case, the tool of choice is KiCad; it creates circuit diagrams and layouts that can be edited by anyone without any major financial outlay. By the way, Elektor published the book *KiCad Like a Pro* for KiCad [7].

The finished circuit board can be seen in **Figure 14** and, as you can see, isn't very beautiful. Cut outs in the board provide the necessary isolation between areas of the board carrying mains voltage and the 3.3 V supply. Any copper superfluous to component interconnection is removed. The result of the routing can be seen in **Figure 15**. Any cut outs or slots in the board outline are made using a milling machine so you cannot expect them to have sharp 90°

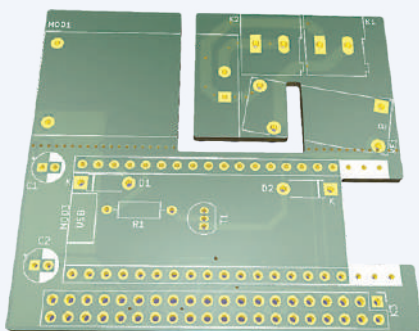


Figure 14: The finished PCB.

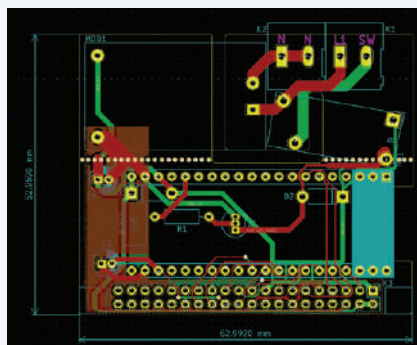


Figure 15: The finished Layout.

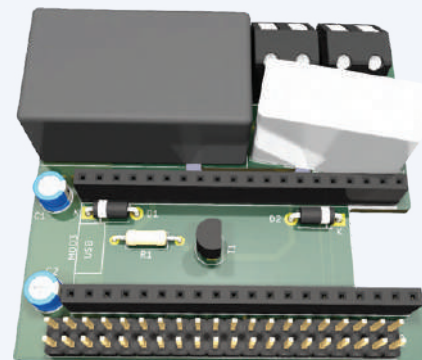


Figure 16: Component layout on the PCB.

corners. If you plan to put the board out for manufacture check the milling process characteristics to find the minimum radius of internal corners.

3D models are now available for almost every component, they can be used to experiment with board layout to fit them in the space available. You may discover you need additional boards, mounted at right angles to the main board, to get everything in. A 3D view of our circuit board can be seen in **Figure 16**.

Shoehorn Please

The nice thing is that we can export a STEP file from KiCad in order to use it in a 3D design program. For this, the model of our circuit board and its housing (3D print in **Figure 17**) are loaded into FreeCAD and placed appropriately. As you can see in **Figure 18**, the circuit board will not fit; the board outline is bigger than the space available, components are shown conflicting with the internal

surfaces of the enclosure and the board edge in green is sticking out of the front. Are the imported 3D models correct? Has anything changed in the scaling? Maybe it's just a measurement error. The answer is the latter.

After adjustments were made to the board dimensions, it all starts to look a little cramped on the circuit board. An edge view of the component placement in **Figure 19** shows there is no room to fit all the components. We need to maintain safe separation between tracks so it will not be possible to move the components closer together. When everything is put together using FreeCAD, **Figure 20** shows the power supply module outline still eats into one internal corner pillar. We will need to go back and rethink the design using alternative components.

What Now?

From an electrical hardware and firmware standpoint the basic

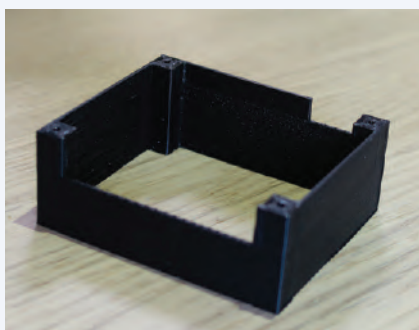


Figure 17: The 3D-printed enclosure.

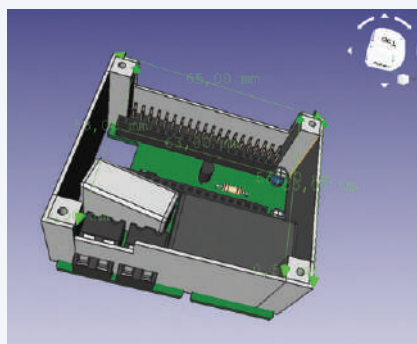


Figure 18: The PCB will not fit.

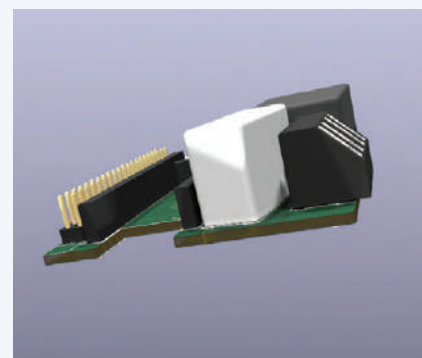


Figure 19: Edge view shows all the components squeezed together.

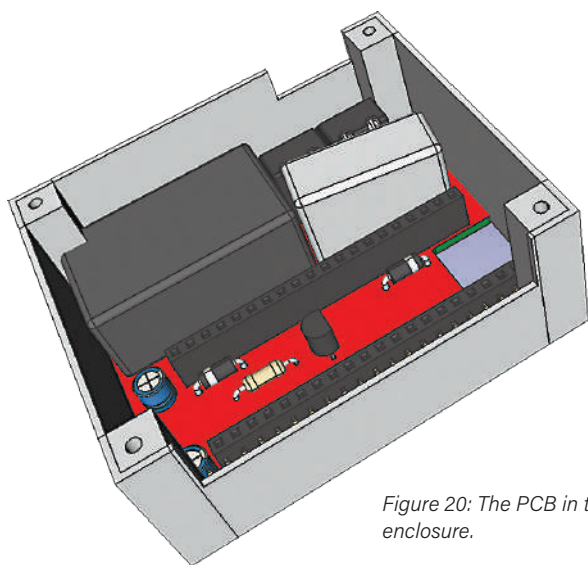



Figure 20: The PCB in the enclosure.

design of the contactless switch works as expected. To make the unit suitable for use across a range of different electrical standards the board should also be small enough to fit inside a circular box. Figure 4 shows a dry-lining box popular in some European countries for flush mounting electrical switches and outlets into plasterboard walls. To resolve this problem, we can look at more compact power supply modules and use a smaller version of the ESP32 module to take up less space than the ESP32 PICO-KIT.

With any luck, we should see a couple of ESP32-C3 modules on the lab bench in the next few days. These modules are more compact, cheaper, come with Wi-Fi and Bluetooth and consume less power. The lower power requirement would mean we could use a smaller power supply module and save a bit more space.

At this point in its development we have decided to temporarily sideline the project. We look to the expertise of our readers and welcome any feedback suggesting alternative, more compact components or how the layout and routing can be optimized to fill the available space. The project is not complete and we expect readers will find many errors in the circuit as it stands. Any hints or tricks on how we can more easily interlink and work

with KiCad and FreeCAD would also be welcome. If you want to take a look at the source code, PCB, circuit diagram or housing data, you can download the project data from the Elektor GitHub repository [8]. 

200478-01

Contributors

Text and images: **Mathias Claußen**

Editors: **Jens Nickel, C. J. Abate**

Translation: **Martin Cooke**

Layout: **Giel Dols**

Questions or Comments?

Do you have any technical questions or comments prompted by this article? Email the author at mathias.claussen@elektor.com or contact Elektor at editor@elektor.com.



RELATED PRODUCTS

- **ESP32-PICO-Kit V4 (SKU 18423)**
www.elektor.com/18423
- **PeakTech 3445 True RMS Digital Multimeter with Bluetooth (6000 Counts) (SKU 18774)**
www.elektor.com/18774
- **Weller WT 1013 Digital soldering station (95 W) (SKU 19338)**
www.elektor.com/19338

WEB LINKS

- [1] Elektor Store: <https://www.elektor.com/new/new-in-the-store>
- [2] Open Smart Switch: <https://www.hackster.io/133854/open-smart-switch-44fa54#toc-future-developments-8>
- [3] You tube video demonstrating Gesture Pad: https://www.youtube.com/watch?v=WVSVhEeMi_4
- [4] MGC3130 Product Page: <https://www.microchip.com/wwwproducts/en/MGC3130>
- [5] MGC3130 Data Sheet: <https://ww1.microchip.com/downloads/en/DeviceDoc/40001718E.pdf>
- [6] Seeed Studio MGC3130 Library: https://github.com/Seeed-Studio/Seeed_Linux_mgc3x30
- [7] Peter Dalmaris, KiCAD like a Pro, Elektor: <http://www.elektor.com/kicad-like-a-pro>
- [8] Github Repository: <https://github.com/ElektorLabs/200478-Touchless-Lightswitch>
- [9] Cavity Wall Device Box by Würth:
<https://eshop.wuerth.de/Produktkategorien/Hohlwand-Geraetedose/14015502030401.cyid/1401.cgid>

Starting Out in Electronics

Matching and Transforming

By Eric Bogers (Elektor)

With this installment, we will bring the “coils” chapter to a close. We will take a final look at crossover filters and discuss the most important aspects of transformers.

Impedance Matching

We briefly return to the band-pass filter that was used in wrapping up the previous instalment [1] (see **Figure 1**). You will notice that we didn't draw a loudspeaker as the load, but used a resistor; because only with a terminating resistor that has a constant value will the filter behave ‘according to the book’ – and the impedance

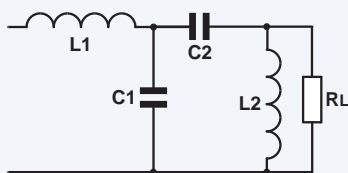


Figure 1: Band-pass filter.

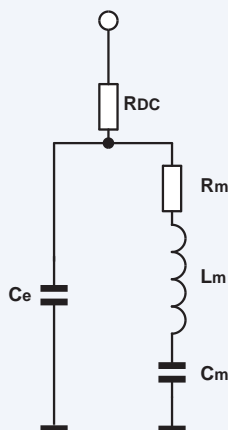
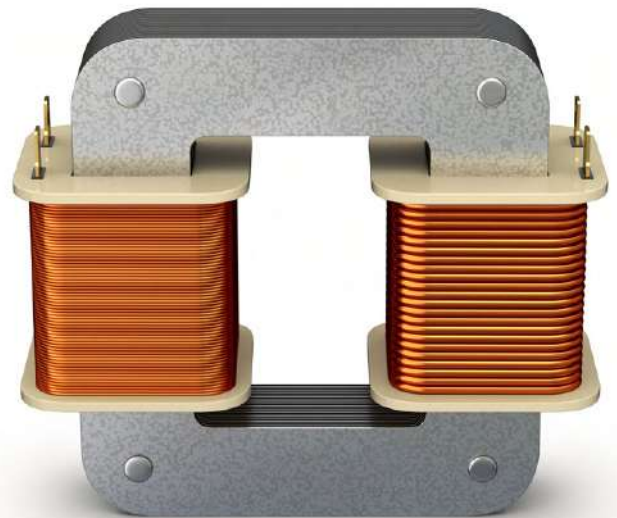


Figure 2: Impedance compensation.



of a loudspeaker is anything but constant. Two factors throw a spanner in the works: firstly, the impedance peaks at the resonance frequency and secondly, the impedance increases with increasing frequency (as a consequence of the self-inductance of the voice coil).

When such disturbances occur at least two octaves away from the crossover frequency of the filter (corresponding to a doubling or halving of the frequency), then you do not have to worry much about this. If, however, this is not the case, then the initial assumptions that were made when calculating the filter are not valid and the filter will not function as intended. Fortunately, it is possible to compensate for this. **Figure 2** shows a practicable compensation network.

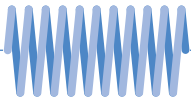
This network has two parts: on the left is capacitor C_e , which has to compensate for the self-inductance of the voice coil, and on the right is a series-resonant circuit that compensates for the resonance peak. When a disturbance factor is sufficiently far from the crossover frequency, then it is not necessary to compensate for that and the relevant part of the network can be omitted.

The values of the components are determined as follows:

R_{DC} = DC resistance of the loudspeaker

$$C_e = \frac{1000 \cdot L_e}{R_{DC}^2} \quad [\mu\text{F}, \text{mH}]$$

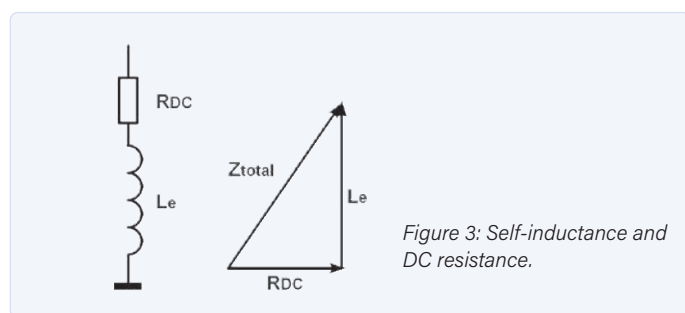
$$R_m = \frac{R_{DC} \cdot Q_e}{Q_m} - R_{loss}$$



$$L_m = \frac{159 \cdot R_{DC} \cdot Q_e}{f_s} \quad [\text{mH}]$$

$$C_m = \frac{159000}{f_s \cdot R_{DC} \cdot Q_e} \quad [\mu\text{F}]$$

The values of the parameters R_{DC} , Q_m , Q_e and f_s can be determined using special measuring programs such as Kirchner ATB. R_{DC} is the DC resistance of the loudspeaker and f_s the centre frequency of the resonance peak. R_{loss} is the equivalent resistance of the coil and L_e its self-inductance.



But how do we determine L_e ? At higher frequencies the loudspeaker can be considered a series circuit of a self-inductance and a DC resistance, as is sketched in **Figure 3**. At a sufficiently high frequency, you can read the value of the total impedance Z_{total} from the impedance diagram. For the self-inductance L_e the following applies:

$$L_e = \frac{\sqrt{Z_{tot}^2 - R_{DC}^2}}{2 \cdot \pi \cdot f}$$

In the example of Figure 3, the total impedance at 10 kHz is 42 Ω , while R_{DC} is equal to 7 Ω . We calculate that a self-inductance of 0.66 mH has to be compensated with a capacitor of about 10 μF .

Transformers

Transformers can be used to transform AC voltages up or down. When mains voltages are transformed down, we normally speak of a mains transformer; when signal voltages need to be transformed, we speak of (you guessed it already) signal transformers. The operation of these two 'species' of transformer are exactly the same however.

Signal transformers are not commonly used for transforming signal voltages up or down but are more typically used to separate (sub-) circuits galvanically from each other — for example, to prevent an earth loop from being created. In such cases we speak of a 'balanced transformer' connection.

Figure 4 shows a few representatives of the genus 'transformer'. On the right, part of a toroidal transformer is visible; here the windings

are wound around a ring-shaped iron core. This type of transformer stands out for its very small stray field, the reason why they are popular in audio power amplifiers.

At top left you can see a printed circuit board transformer — this is a transformer that can be soldered directly into a circuit board. For isolation and to prevent mechanical damage, this transformer is moulded into a plastic enclosure.

Finally, at bottom left we see two audio signal transformers. These too are for circuit board mounting. Audio transformers often have a shield made from tin sheet to prevent them picking up stray magnetic fields. This is, however, not the case in the examples of Figure 4.

In **Figure 5** you will see the schematic symbol for a transformer. If this makes you think of two coils, then you are right: a transformer consists of two (or more) coils that are wound on a common core. In principle that core could be a big chunk of iron, but that would lead to unacceptably high AC losses. That is why in 50-Hz transformers (i.e., mains transformers), the core consists of thin sheets of transformer steel, which are insulated from each other. For higher frequencies, iron powder is mixed with an isolating filler and pressed into the desired shape. This is how ferrite cores are obtained. At very high frequencies a core is no longer required and you can use two air-cored coils.

A transformer operates as follows: an AC voltage is applied to the primary winding; there will then flow an AC current through the winding, which generates a varying magnetic field. This varying magnetic field then induces an AC voltage in the secondary winding. There is no electrically conducting path between the primary and secondary windings: they are galvanically separated.

The names 'primary' and 'secondary' might lead to the impression that a transformer operates in only one direction — but nothing is further from the truth: it is certainly possible to apply a voltage to the secondary winding and subsequently use the transformed voltage on the primary winding. However, when doing this, we have to make sure that the iron core is not driven into (magnetic) saturation by a voltage that is too high. If, for example, a 12-V transformer (that is, a transformer that is normally used to transform the 230-V mains voltage down to 12V) is connected backwards, then a blown fuse (or a burnt out transformer winding) is a guaranteed outcome. As a consequence of the magnetic saturation of the core, a current that is much too great will flow. It is, however, perfectly possible to apply a voltage of 12 V on the secondary winding so that 230 V is available from the primary winding. The so-called inverters operate according to this principle.

The ratio between the number of primary turns and the number of secondary turns is called the transformer turns ratio. The following applies:

$$T = \frac{W_1}{W_2} = \frac{U_1}{U_2} = \frac{I_2}{I_1}$$

The ratio between the input and output voltages is equal to the ratio of the number of primary and secondary turns: the ratio of the input and output currents is exactly the opposite. This is obvious, of course, because the following must be true:

$$P_1 = P_2$$


The power in the secondary winding is not exactly the same magnitude as that in the primary winding. A transformer is not an ideal component so that there is always some amount of loss. However, these losses are typically so small that in most calculations we can ignore them.

For mains transformers, no words are normally wasted about the transformer turns ratio: here only the voltage is indicated that will appear on the secondary winding when the mains voltage of 230 V is applied to the primary winding.

Many mains transformers have multiple secondary windings, sometimes with different output voltages. It is possible to connect secondary windings in series, but we have to take phase and maximum load (maximum current) into account. If we want to connect multiple windings in parallel, then these must supply the same voltage and have a (nearly) identical power capability. Furthermore, the phases of the parallel-connected windings have to be equal.

In addition, mains transformers often have multiple primary windings. This is done so that the devices that have them built in can be adapted to the different mains voltages in different countries.

It is however never possible to connect a 230-V mains transformer to a voltage of 400 V: the transformer will go into magnetic saturation which will result in a blown fuse or the primary winding itself will burn out. This is the reason why wrongly installed three-phase outlets are feared everywhere. The opposite is, however, perfectly okay. A 400-V transformer can be built into electronic equipment; these have no problems operating from 230 V, but do supply, of course, a correspondingly lower output voltage.

This concludes the subject of "coils." Next time, we will (finally!) start to deal with semiconductors. 

210626-01

The series of articles "Starting out in electronics" is based on the book, Basic Electronics Course, by Michael Ebner, published by Elektor.

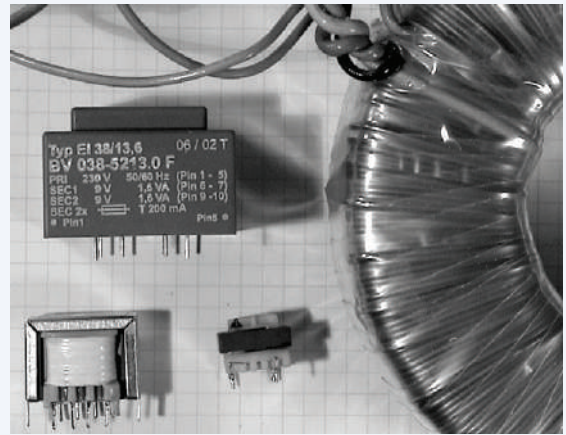


Figure 4: Various mains and signal transformers.

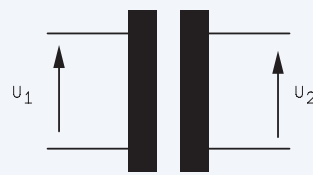


Figure 5:
Schematic symbol
for a transformer.

Questions or Comments?

Do you have any technical questions or comments prompted by this article? Send an email to the author or to the editor of Elektor via editor@elektor.com.

Contributors

Idea and illustrations: Michael Ebner

Text and editing: Eric Bogers

Translation: Arthur de Beun

Layout: Giel Dols



RELATED PRODUCTS

> **B. Kainka, Basic Electronics for Beginners, Elektor 2020. (SKU 19212)**
www.elektor.com/13950

> **B. Kainka, Basic Electronics for Beginners (E-Book), Elektor 2020. (SKU 19213)**
www.elektor.com/18232

WEB LINK

[1] "Starting out in electronics," Elektor Magazine January/February 2022: <http://www.elektormagazine.com/210564-01>

What's New in Embedded Development?

Rust and Keeping IoT Deployments Updated



By **Stuart Cording** (Elektor)

While the products embedded systems engineers release to the market make it seem like technology is moving forward rapidly, the industry itself is, by comparison, slow. That's why it was such a shock when Raspberry Pi, the renowned single-board computing creator, brought out the RP2040 microcontroller (MCU) with its dual Cortex-M0+ cores and no onboard flash. Dual-core in this class of devices is unheard of. But that is about as exciting as it has got. Progress in the world of embedded systems is otherwise measured, meaningful, and considered. But several developments are underway that could change embedded development in the decade ahead, as we shall see.

Embedded software development without C is almost impossible to imagine. As assembler became too cumbersome to develop entire applications, C displaced it except in specific cases when highly-optimized, hand-coded assembler was the only option. The language, developed by Dennis Ritchie [1] at Bell Labs, provides enough flexibility to develop complex applications while also providing easy access to registers. This is critical to writing compact microcontroller code that handles register accesses in interrupt routines. It is also easy to implement tasks such as bit manipulation of registers. And, unlike code written in assembler, the resultant code is easier to read. C also ranks consistently as a preferred programming language, ranking in the top three programming languages in surveys and market analyses (**Figure 1**) [2][3].

C Is Old

However, C, developed in 1972, is now 50 years old. It has a range of limitations that are well known, many of which relate to the use of pointers. While pointers make it easy for embedded developers to access registers, they can also result in unwanted out-of-bounds memory accesses. Additionally, compared to more modern programming languages, C compilers undertake comparatively few code checks. As a result, unused variables are simply ignored, something which could signify a coding mistake.

To ensure unsafe C code is not integrated into embedded systems, developers use coding standards such as MISRA C [4]. This standard came about as C grew in importance in the automotive industry as a programming language for embedded systems. C++ resolves some

of C's issues with pointers with *references* [5] that cannot be changed to refer to another object, cannot be NULL, and must be initialized on creation. Despite this, AUTOSAR, a development partnership of automotive systems developers, developed guidelines for the usage of C++ for safety-related applications in a document with several hundred pages [6]. So, while competency in these established languages is essential for embedded developers, it is clear that each language has enough shortcomings that guidelines are needed to avoid common programming failures.

Introducing Rust

Rust has emerged as a potential contender, touting itself as highly suited for developing safe systems. It started as a private project by Graydon Hoare in 2006, becoming a sponsored project of his employer, Mozilla Research, around 2010. In 2021, the Rust Foundation [7] was formed after company restructuring impacted the Rust development team.

What makes Rust different is that, at compile time, many issues are caught and highlighted, often for issues that C/C++ compilers would ignore. An *ownership* system for variable declarations is also implemented. This uses a *borrow-checker* to avoid the misuse of variables with enforcement during compilation. Additionally, read and write access to variables passed by reference must be explicitly declared. Much of Rust looks syntactically similar to C and C++, using curly brackets around functions and the well-known control keywords.

Due to its focus on safe code, work has been undertaken to deliver Rust to the bare metal embedded software development community. However, due to the nature of embedded programming, the static

checking employed by the compiler can cause issues when implementing some types of code. Some code sections can be marked *unsafe* to allow, for example, pointer dereferencing to circumvent this. The thinking here is that, by explicitly defining code sections as unsafe, the code clarifies circumvention of the rules of Rust.

Taking Rust for a Spin

One of the best starting points to get a feel for Rust is with the Raspberry Pi. Installation is simple, following the guidelines at the website rustup.rs [8]. From the command line, simply enter the following command and follow the instructions provided:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Unlike C/C++ that generates binaries, the output of RUST is known as a *crate*. The package manager Cargo is used to simplify the command line compilation process. This stores the data needed to generate the crate and allows the developer to define the packages required to build it. By invoking Cargo from the command line, a new Rust project is generated as follows:

```
cargo new rust_test_project
```

Opening the new project's directory shows a file named *Cargo.toml*. This file must be modified as required. For a simple Raspberry Pi application that blinks an LED connected to a GPIO, a suitable crate dependency has to be defined to access the GPIO pins. Crates are shared on crates.io [9], a platform dedicated to the Rust community's crates. A

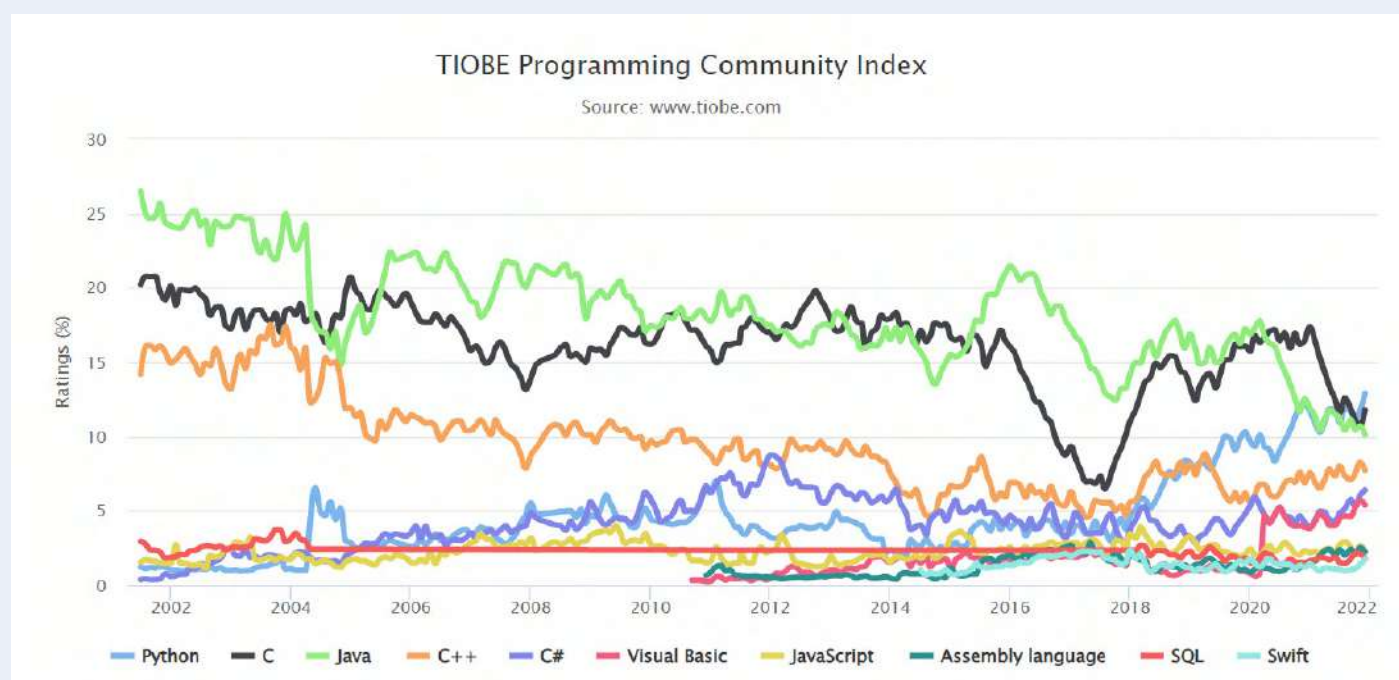


Figure 1: C remains popular as a programming language, regularly placing in the top three in surveys and market analyses. (Source: www.tiobe.com)

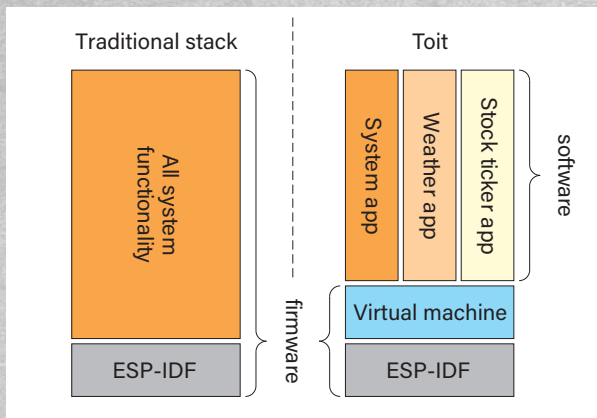


Figure 2: Toit executes IoT code as apps on top of a virtual machine running on an ESP32. (Source: Toit)

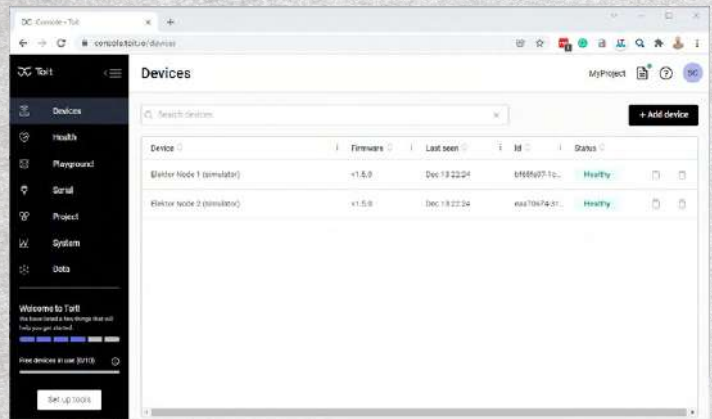


Figure 3: The Toit Console in a browser, here displaying two simulated ESP32 nodes.

suitable crate is *rppl*, which can be found via the search feature. The platform shows whether the crate can be built currently, provides the version number, and offers documentation and example code. The crate name and version number are then added as *dependencies* in *Cargo.toml* (Listing 1).



Listing 1: Adding the *rppl* dependency to *Cargo.toml* in a Rust project.

```
[package]
name = "rust_test_project"
version = "0.1.0"
edition = "2021"
[dependencies]
rppl = "0.13.1"
```

Moving to the *src* directory, the developer finds a simple *Hello World* example project in the Rust source code file *main.rs*. Replacing this code with Listing 2 allows an LED connected to GPIO 23 (pin 16 of the Raspberry Pi header) to be flashed ten times. Compilation is undertaken from the command line using *cargo build*, and the crate is executed using *cargo run*.

Something holding back the use of Rust on some microcontrollers is the need to use an LLVM toolchain. If this is available, you can get started using one of the various online tutorials. One example is provided for the BBC micro:bit [10] and shows that, once the Rust syntax has been understood (Listing 3 [11]), it's very similar to writing code in C/C++. Another example for the STM32 [12] is also provided.

Is Rust the Future?

So, what is holding back the adoption of Rust for embedded? Well, if you're looking for a robust programming language suited for use in safety-critical systems, Ada [13] already has you covered. And,



Listing 2: Blinking an LED in Rust.

Blinking an LED in Rust is similar to code written in C. This example is based upon code included with the *rppl* crate.

```
use std::error::Error;
use std::thread;
use std::time::Duration;
use rppl::gpio::Gpio;
use rppl::system::DeviceInfo;
// Gpio uses BCM pin numbering. BCM GPIO 23 is tied to physical pin 16.
const GPIO_LED: u8 = 23;
fn main() -> Result<(), Box<dyn Error>> {
    let mut n = 1;
    println!("Blinking an LED on a {}. ", DeviceInfo::new()?.model());
    let mut pin = Gpio::new()?.get(GPIO_LED)?.into_output();
    while n < 11 {
        // Blink the LED by setting the pin's logic level high for 500 ms.
        pin.set_high();
        thread::sleep(Duration::from_millis(500));
        pin.set_low();
        thread::sleep(Duration::from_millis(500));
        n += 1;
    }
    Ok(())
}
```


despite being 40 years old, it also hasn't yet managed to displace C, despite being developed specifically for use in real-time embedded systems. Another aspect is C's years of success, with countless developers, tools, and libraries of code available.

However, the availability of the Crate package manager could help speed up its adoption. Keeping track of semiconductor vendor peripheral libraries written in C/C++ can be challenging and opaque, so the explicit definition of the version of the crates used in [Cargo.toml](#) could be seen as a significant improvement. It may also simplify the lives of MCU manufacturers trying to support their massive portfolios of products. However, Ada has already responded with the release in 2020 of their *Alire* package manager [14]. And, just like for Rust, Ada already includes support for the BBC micro:bit, should you wish to compare the two languages with an MCU [15].

Keeping IoT Devices Up-to-Date

With ever more embedded devices connected to networks, the biggest challenge is keeping them updated with the most recent version of the firmware. Traditionally, firmware updates have to be downloaded over the air with the binary programmed into flash using an on-chip bootloader stored in a protected area of the device. However, the risk is that this new code version fails to deploy correctly, bricking the product and making it unusable. Alternatively, the code may function, but a software bug in the new code could hinder future updates from deploying, leaving vulnerable devices in the field. Thus, despite the majority of the application functioning correctly, the device becomes unusable, perhaps due to a mistake in a single line of code.

For years, virtual machines have been a standard method for deploying multiple apps or operating systems on servers. The virtual machine allows an operating system to be executed, believing it is on dedicated hardware. Should the operating system fail catastrophically, only the affected virtual machine is impacted, not the remaining systems running on the server. Desktop users will be aware of virtualization software such as VirtualBox, VMware, and Parallels, allowing them to trial software or alternate operating systems without placing their primary machine at any risk.

Updating IoT Node Firmware: Getting Around Toit

The team at Toit, a Danish company, formed by ex-Google engineers, wondered why virtualization wasn't in use on microcontrollers running IoT applications. After all, they require regular updates to ensure bugs are fixed and may even need changes to support improvements made to the cloud services they communicate with. And, obviously, no one wants to manually flash IoT nodes in the field if the devices get bricked by an update.

To get started, they opted for the ESP32 from Espressif. The recommended hardware is the ESP32-WROOM-32E with a dual-core 32-bit Xtensa LX6 microprocessor, 520 KB SRAM, and 4 MB of flash. The



Listing 3: Simplified code snippet that checks the state of an input pin on the BBC micro:bit using Rust.

```
#![no_std]
#![no_main]

extern crate panic_abort;
extern crate cortex_m_rt as rt;
extern crate microbit;

use rt::entry;
use microbit::hal::prelude::*;

#[entry]
fn main() -> ! {
    if let Some(p) = microbit::Peripherals::take() {
        // Split GPIO
        let mut gpio = p.GPIO.split();

        // Configure button GPIO as input
        let button_a = gpio.pin17.into_floating_input();

        // loop variable
        let mut state_a_low = false;

        loop {
            // Get button state
            let button_a_low = button_a.is_low();

            if button_a_low && !state_a_low {
                // Output message
            }

            if !button_a_low && state_a_low {
                // Output message
            }

            // Store button states
            state_a_low = button_a_low;
        }
    }
    panic!("End");
}
```

platform already has its ESP-IDF (Espressif IoT Development Framework), so it's ready for use as an IoT node. Toit has then built a virtual machine (**Figure 2**) that allows apps to execute safely on top. Apps are written in Toit, a high-level, object-oriented, and safe language.

Testing Toit with Simulated ESP32 Nodes

Managing and deploying apps across a collection of IoT nodes is handled using the Toit Console coupled with Microsoft's Visual Studio Code. For those only interested in trialing the platform, the Toit Console allows simulated ESP32 devices to be used. After installing the Toit extension for Visual Studio Code, apps can be written and simulated locally or deployed to devices attached to the Console.

A YAML file accompanies Toit apps. This markup language file is used to declare the Toit app name, source code file, and define triggers. These can execute the app after booting, installation, and set time intervals.

Deployment of apps or updates does not require the target ESP32 device to be disabled, or power cycled. Instead, the virtual machine

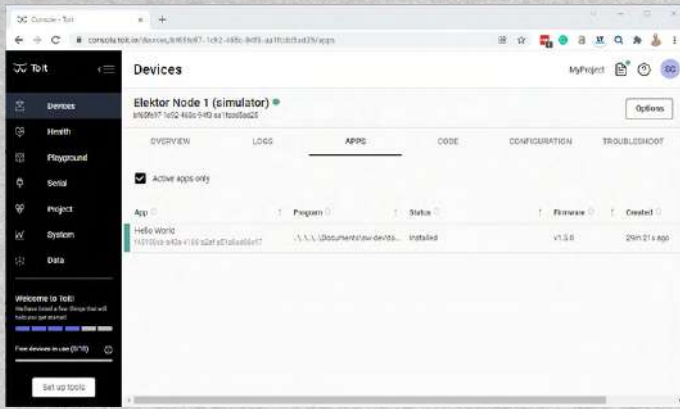


Figure 4: The Hello World app successfully installed on one simulated node.

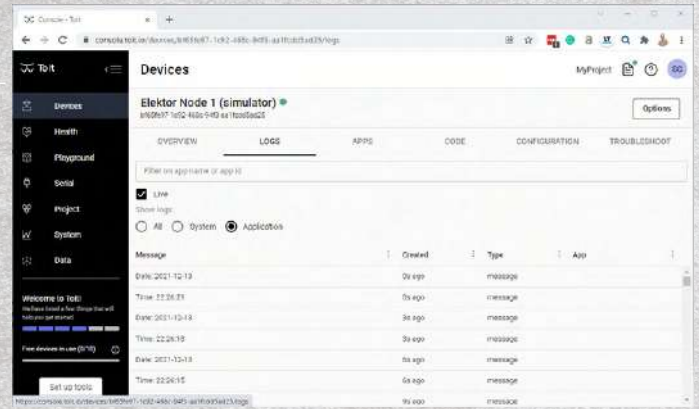


Figure 5: The LOGS output shows the time and date being output from the Toit app every three seconds, as defined in the YAML file.

updates the app in-situ and triggers it as per the settings provided in the YAML file. If the app fails somehow, such as through a stack overflow, the remaining apps continue executing without issue [16]. Such errors can be diagnosed through the Console by reviewing the log.

A simple *Hello World* app that outputs the time and date is shown in **Listing 4**, accompanied by its YAML file in **Listing 5**. **Figure 3** shows two simulated nodes in the Console, while **Figure 4** shows the *Hello World* app successfully installed. Finally, **Figure 5** shows the date and time output at three-second intervals as defined using the `on_interval: "3s"` trigger in the YAML file. The project code is created in Visual Studio Code and, using the Toit extension, deployed to the chosen node attached to the user's Console account (**Figure 6**).

At first glance, the Toit approach may seem a little restrictive. The virtualization environment only allows control of the GPIOs, serial interface (UART), SPI, or I²C. However, with many IoT nodes collecting sensor data and reporting these to the cloud, this seems to provide enough flexibility for most applications. Toit also operates its own package manager (registry) [17], providing drivers for a range of sensors, input devices, LCDs, and other utilities. The environment

also supports the low-power mode of the ESP32, which they claim allows the device to run on two AA batteries for years [18]. If an app update occurs while the IoT node is in deep-sleep mode, the Console deploys it when the node next wakes up.

Rust and Toit — The Future of Embedded?

Two things stand out with both Rust and Toit. Both are simple to get up and running, and both use package managers to handle the low-level drivers. This allows developers to focus on their job — building applications. With applications becoming increasingly complex, such code reuse is essential to reduce development time and bring products to market faster.

Rust has a huge mountain to climb. With C and C++ so entrenched in the world of embedded development, it is difficult to find a chink in the armor where it would deliver a benefit significant enough to draw developers in. And, with Ada already established as *the* language for safety-critical systems, Rust's advantages essentially disappear.

Toit, by contrast, solves a major headache: keeping remote IoT nodes up-to-date, deploying new features to the user base, and all without



Listing 4: A simple app written in Toit that outputs formatted time and date strings to the log in Console.

```
main:
  time := Time.now.local
  print "Time: ${%02d time.h}:${%02d time.m}:${%02d time.s}"
  print "Date: ${%04d time.year}-${%02d time.month}-${%02d time.day}"
```



Listing 5: The accompanying YAML file that tells the Toit Console how to deploy the app.

```
name: Hello World
entrypoint: hello_world.toit
triggers:
  on_boot: true
  on_install: true
  on_interval: "3s"
```

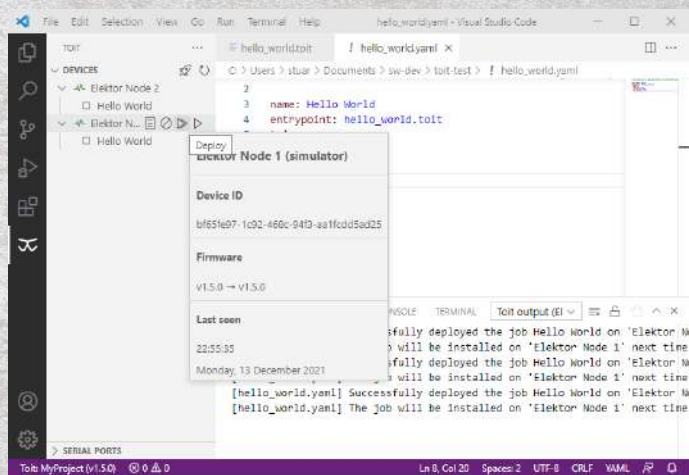



Figure 6: Using the Toit extension in Visual Studio Code, changes to apps can be deployed immediately to IoT nodes without fear of bricking devices in the field.

bricking devices out in the field. Some developers will be concerned by the 4 MB minimum flash requirement and that, currently, only ESP32 is supported. However, should the market demand appear for other MCUs, there doesn't seem to be a technical reason why other platforms wouldn't be supported in the future. 

210652-01

Questions or Comments?

Do you have technical questions or comments about this article? If so, please contact the author at stuart.cording@elektor.com or the Elektor editorial staff at editor@elektor.com.

Contributors

Text/Images: **Stuart Cording**
 Editors: **Jens Nickel, CJ Abate**
 Layout: **Harmen Heida**



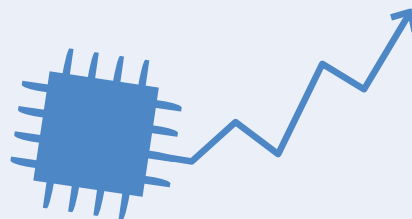
RELATED PRODUCTS

- **D. Ibrahim, BBC micro:bit (Elektor, 2016, SKU 17972)**
www.elektor.com/17872
- **Joy-IT BBC micro:bit Go Set (SKU 18930)**
www.elektor.com/18930

WEB LINKS

- [1] Dennis Ritchie, Computer History Museum: <https://bit.ly/3oOXG1A>
- [2] P. Jansen, "TIOBE Index for December 2021," TIOBE Software BV, December 2021: <https://bit.ly/3EXx8Ri>
- [3] S. Cass, "Top Programming Languages 2021," IEEE Spectrum, 2021: <https://bit.ly/3oPjq8z>
- [4] MISRA Website: <https://bit.ly/3s1Mv7D>
- [5] "C++ References," Tutorials Point: <https://bit.ly/31Y6k53>
- [6] "Guidelines for the use of the C++14 language in critical and safety-related systems," AUTOSAR, October 2018: <https://bit.ly/3dO3zWl>
- [7] Rust Foundation Website: <https://bit.ly/3DNgO45>
- [8] rustup Website: <https://bit.ly/3EUTiDR>
- [9] Rust Crate Registry Website: <https://bit.ly/3dLKmor>
- [10] droogmic, "MicroRust," April 2020: <https://bit.ly/3EUWFdM>
- [11] droogmic, "MicroRust: Buttons," April 2020: <https://bit.ly/3DWes30>
- [12] bors and NitinSaxenait, "Discovery," December 2021: <http://bit.ly/3GERDT7>
- [13] Get Ada Now Website: <https://bit.ly/3GHyikw>
- [14] F. Chouteau, "First beta release of Alire, the package manager for Ada/SPARK," AdaCore, October 2020: <https://bit.ly/3EUXOSC>
- [15] F. Chouteau, "Microbit_examples," December 2021: <https://bit.ly/3mnH7bx>
- [16] "Watch us turn an ESP32 into a full computer!," Toit, March 2021: <https://bit.ly/3IM0WT8>
- [17] Toit Package Registry Website: <https://bit.ly/3yokpo7>
- [18] Toit Docs: Prerequisites, Website: <https://bit.ly/3oNSECq>

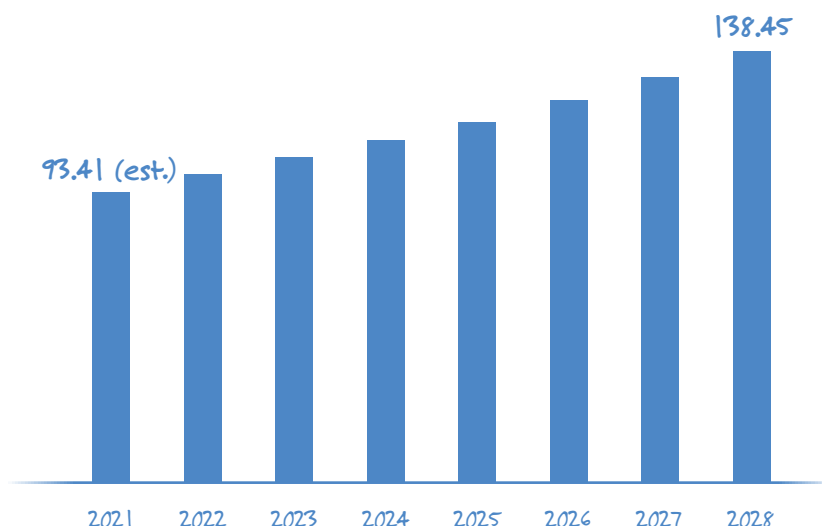
Embedded: Its Growth Is Solidly Embedded Indeed



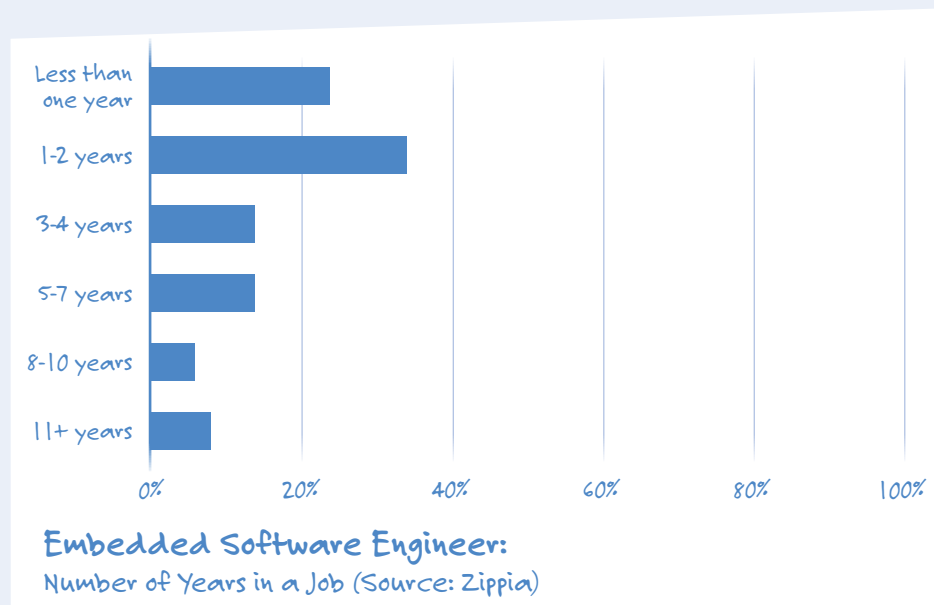
According to market research company The Brainy Insights, the global market for embedded systems still looks very promising, as if Covid-19 has never been around. Its research leads to a compound annual growth rate (CAGR) of 5.73% for the years 2021 to 2028. Although this estimate is rather cautious compared to some other market research firms, the report of the Indian research company has a very optimistic undertone indeed. Automotive (market share 5%), Healthcare (10%) and Communications (45%) are promising sectors within the embedded club and already make up 60% of its revenue.

(Sources: The Brainy Insights, The Insight Partners)

Global Embedded System Market Size, 2021-2028 (\$ Billion)

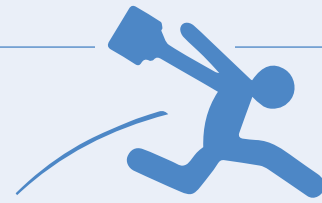


Embedded Software Engineer? You Are in the Driving Seat



Are there restraints holding back some of the embedded market's potential? Absolutely. One of them is the lack of skilled software engineers. Whereas back in 2010 about 4.5% of the embedded software engineers were unemployed (US figures), now that is below 2%. The demand for software engineers will be 21% higher in 2028 than in 2021/2022, according to recruitment company Built In. The average growth for all occupations will be around 5% for the same period. Embedded software engineers can pick and choose what they want, leading to job hopping for 60% of them.

(Sources: Built In, Career research site Zippia)



Hopping from One Job to the Other

Scenario	Cost	Duration
PCB Layout Services with optional Signal Integrity analysis and optimization	~\$5K to \$35K	~1 to 8 weeks
Low to medium complexity microcontroller-based board schematic design and PCB layout and firmware development	~\$25K to \$50K	~6 to 12 weeks
FPGA VHDL development	~\$15K to \$125K	~1 to 6+ months
Firmware/software development	~\$10K to \$125K	~2 weeks to 6+ months

Why can embedded software engineers hop from one job to the other without feeling uncomfortable? That is because the job of designing a typical embedded system roughly takes six months. Have a look at the table set up by embedded

system developer AppliedLogix. This American company has its clients in a wide range of sectors and is qualified to give some averages. These averages make perfectly clear that an embedded software engineer can complete a job,

feel good about it and turn to the next assignment inside or outside the same company.

(Sources: AppliedLogix, Zippia)

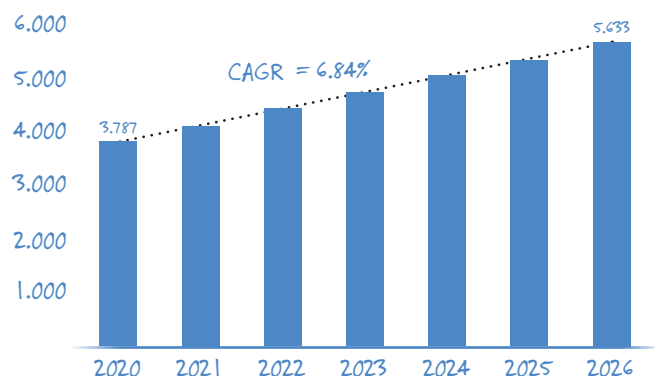
8-Bit MCUs Falling Somewhat Behind

Although 8-bit microcontrollers are here to stay, even at the end of this decade, it is also true that the 32-bit MCUs are making more headway than originally expected. The fact that 32-bit controllers are gaining ground most likely has to do with the fast growth of real-time embedded systems. Whereas embedded systems as a whole will grow 5.7% annually between now and 2028, real-time embedded systems will roughly grow 1 percentage point more (6.9%). Also, the unit price of 32-bit MCUs have been declining steadily. The easy way of thinking (25% 8-bit, 25% 32-bit, 50% 16-bit) belongs to the past.

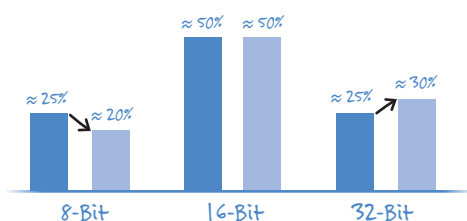
(Sources: Allied Market Research, The Brainy Insights, Grand View Research)

Global Embedded Security Market, 2020-2026 (\$ Billion)

(Source: Knowledge Sourcing Intelligence)



Market Share Microcontrollers, 2021 vs. 2027



Security: Is It Taken Care Of?



The last time we made an Infographics on security solutions for embedded systems, two years ago, it became apparent that engineers were lacking behind in implementing these solutions. Is this still the case? No, it is not. The market for security solutions for embedded systems is growing faster than the market for embedded systems itself. The difference in growth rate is not spectacular — 5.73% against 6.84% during the coming years — but it is enough not to be alarmed any longer. Not only software protection is being implemented, also hardware identification.

(Sources: The Brainy Insights, Knowledge Sourcing Intelligence)



How the Industrial and Automotive Sectors Will Benefit from 5G

By Mark Patrick (Mouser Electronics)

5G is about more than improving mobile telephony. The higher download speeds may enhance the browsing experience on a smartphone, but the real impact of 5G is more likely to come from applications that are yet to emerge. Let's look at how 5G may impact the industrial and automotive sectors.

Why 5G in the Industrial Manufacturing Sector?

Many of today's smart factories are constrained by the limitations of existing wired architectures, which use proven networks like Industrial Ethernet, Profinet, and CANbus to connect the various sensors, actuators, and controllers found in automated equipment. This hard-wired connectivity makes even small modifications to production facilities time-consuming and costly.

Previous generations of wireless networks, including the faster 4G/LTE, have been unable to deliver the real-time responsiveness and low latency required for autonomy. Also, the factory floor is a difficult operating environment, with high levels of electrical noise and interference challenging the performance of many previous wireless communications technologies. 5G's enhanced networking capabilities can address some of these issues, increasing system efficiency and flexibility.

One of the key functions of any automated factory is monitoring. 5G brings Massive Machine-Type Communications (mMTC) capability, which fulfils the needs of extensive wireless sensor networks (WSN). 5G is also more energy efficient than its predecessors, which is critical for extending the battery life of these connected devices, thereby minimising maintenance.

For motion control and industrial robotics, which require precision and real-time sensitivity, Time-Sensitive Networking (TSN) using wired Industrial Ethernet has been the preferred network technology. With its Ultra-Reliable Low-Latency Communication (URLLC), 5G is a viable wireless alternative and additionally enables cloud robotics.

Three related technologies that are emerging into the factory environment are Virtual Reality, Augmented Reality, and Artificial Intelligence (VR/AR/AI). With its high speed and URLLC, 5G enables

processing at the edge. Here, energy-intensive computations can be performed in the cloud, enabling less complex and lower-cost devices on the field side.

5G Brings Challenges as Well as Opportunities

To protect prior investments in previous wired and wireless network technologies, 5G projects must integrate seamlessly into the existing infrastructure. One of the key challenges so far is that indoor coverage has never been a priority for Mobile Network Operators (MNOs). Developments in Open-RAN technologies reduce the cost of ownership of 5G Radio Access Networks (5G RAN), making Private 5G, also known as Non-Public Network (NPN), deployments a realistic possibility. For businesses that prefer this option, regulators worldwide are making a dedicated, cost-effective spectrum available for private 5G. In addition, depending on the operational needs of the factory, private 5G can either be wholly isolated from the public network or shared.

5G and the Era of the Connected Car

The automotive sector is also forecast to be at the leading edge of the 5G roll-out, though it may be a few years before level 5 (L5) autonomy is a commercial reality. It is, however, likely that the next car you buy will be Internet-enabled to manage telemat-

ics, Cellular Vehicle-to-Everything (C-V2X), and infotainment.

Today's connected car can generate as much as 4 TB of data per day, equivalent to about 500 movies. Recent developments in C-V2X communications technology is already using this data in many ways. Data from the engine management systems, for example, is now being sent to remote service centres for predictive maintenance. Information about the local traffic conditions and weather can also feed into Public Safety Systems. Even driver behaviour and vehicle mileage can feed databases for usage-based insurance schemes.

Over the past 5 years, the 3rd Generation Partnership Project (3GPP), which is a global standards body for cellular telecommunications technologies, including radio access, core network and service capabilities, which provide a complete system description for mobile telecommunications, has been increasing the functionality of C-V2X in line with developments in cellular networking technology. The capabilities of Release 16 are paving the way for advanced driver-assistance systems (ADAS).

Although widespread availability of self-driving cars may seem a while away, there have been some very high-profile trials. Tesla, Google, and BMW are all making the headlines, building the general public's expectations and driving momentum. Many high-end vehicles already have some level of autonomy, some up to Level 3 (L3), which also depend on C-V2X technologies.

Although 4G/LTE networks support many of the applications mentioned above, the escalating volume of shared data puts increasing pressure on the available bandwidth. In addition, as critical onboard safety and energy management systems become ever more sophisticated, low latency performance becomes a necessity. The network speeds and cloud-edge processing capabilities must support human-reflex levels of latency to realise higher levels of autonomy. So too, for more sophisticated ADAS, the connected car

About the Author

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



must respond to surrounding events in real-time. The current wireless network is reaching its limit and becoming more of a barrier — without 5G, there will be no self-driving car.

Conclusion

Most of the 5G network roll-out has focused on upgrading 4G/LTE using 3GPP's 5G New Radio Non-Standalone (5G NR NSA), Release 15, specifications, which has enabled the launch of a limited range of 5G services. However, the true potential of 5G is based on the deployment of 3GPP's Release 16 and, further down the road, Release 17. Applications such as the autonomous car and factory

autonomy will only become a reality when they have easy access to this next level of network performance. The initial roll-out of 5G has been somewhat cautious, which has been hampered by the impact of the global pandemic. The second wave of the 5G network roll-out will certainly accelerate demand for a broad spectrum of yet to be discovered applications. ◀

220061-01

For more 5G information, visit Mouser's *Empowering Innovation Together* site: www.mouser.com/empowering-innovation/5G



Moving Coil Relays

Peculiar Parts, the series

By **David Ashton** (Australia)

In this issue, we tackle a really peculiar component. So much so, we've been unable to find out much about it except for its physical properties. But, despite its moving-coil construction, we're pretty sure it isn't a meter!



Say moving coil, and most of you will immediately think of meters. While they're now a bit passé, most people have seen or used them at some time or other. However, I'm talking about moving coil relays. Imagine connecting a wire to the needle of a meter and then placing another contact at the end-stop. As the meter achieves full-scale deflection, it makes contact. That's basically how these relays work.

I found these moving-coil relays in a switchboard some years ago, and it took me some time to work out what they were. They are things of beauty but, in addition to that, they are among the most sensitive relays I have ever come across.

They are made by BBC Goerz Electro. It's challenging to find any information on this company. We do know that Goerz was an Austrian optical company. BBC (Brown Boveri Corporation) was and is a Swiss electrical company, which has been through many iterations to become, as it is known today, ABB (Asea Brown Boveri). BBC Goerz made many items of professional-looking test equipment using the trade name Metrawatt, and Gossen Metrawatt currently makes multimeters and other test equipment. A somewhat chequered career you might say.

Despite that background on the manufacturer, I can find no information on these relays apart from someone selling some on eBay for around \$100 each. They are built on an 8-way octal valve type base. When I first got them, I wondered if they were a type of valve. However, they have a clear plastic case that screws off, allowing you to look at the intricate mechanism inside. The type number, 91041-2, is clearly printed on the case along with a serial number (**Figure 1**). Furthermore, there is a connection diagram on the side, which makes it easier to test them (**Figure 2**).



Figure 1: The relays, shown here from behind, with their type number and plastic case on display.

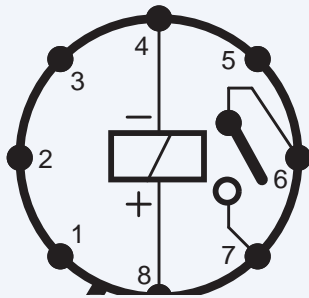


Figure 2: This circuit diagram is printed on the side of the relay.

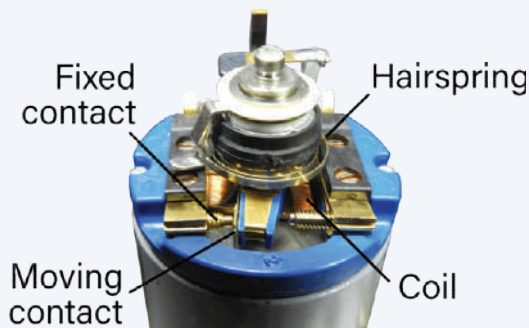



Figure 3: An annotated photo highlighting the location of the hairspring, coil, and contacts.

I rigged up a test jig for these relays — helpfully, I have three sockets on a bracket with plenty of holes to mount other components. I used a 470 k Ω potentiometer in series with the relay coil, together with a 12 V supply, and used the contact to drive a LED so I could see when it was closed. They operate at a current of just 260 μ A at a voltage of 113 mV — show me another relay that sensitive!

The contacts are small (**Figure 3**) and you would have to use them to operate another relay to switch any meaningful power. These days, of course, you'd use an optoisolator to do the same thing, making me think that these devices are pretty old, but you'd still need a bit more electronics to achieve these specifications. These relays are certainly peculiar! 

210557-01

Questions or Comments?

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

Accelerate test with SDG7000A!



SDG7000A

Arbitrary Signal Generator

- 350 MHz/500 MHz/1 GHz
- 2 single-ended/differential Channels
- 16 digital Channels (opt)
- 512 Mpts arb. memory

 **SIGLENT**[®]

www.siglenteu.com

Info-eu@siglent.com

The Elektor Store

Never expensive, always surprising

The Elektor Store has developed from the community store for Elektor's own products like books, magazines, kits and modules, into a mature webshop that offers great value for surprising electronics. We offer the products

that we ourselves are enthusiastic about or that we simply want to try out. If you have a nice suggestion, we are here (sale@elektor.com). Our main conditions:
never expensive, always surprising!

Elektor Archive 1974-2021 (USB Stick)



Price: €199.95

Member Price: €99.95

 www.elektor.com/20071

Six Digit Nixie Clock with IN-14 Tubes

Price: ~~€299.00~~

Special Price: €269.00

 www.elektor.com/20044





Great Scott Gadgets Opera Cake – Antenna Switch for HackRF One



Price: €169.95

Member Price: €152.96

www.elektor.com/20083

Velleman VTSS210 Multifunctional SMD Repair Station

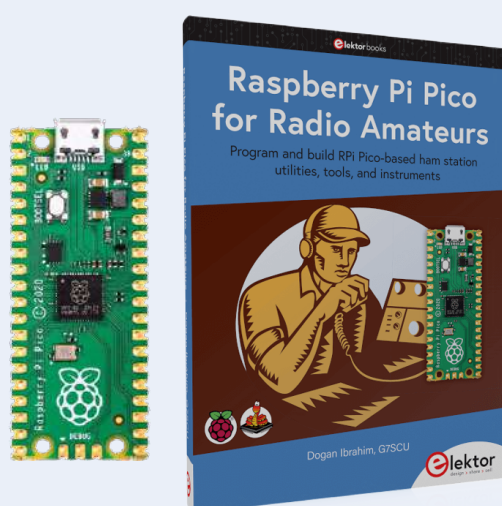


Price: €84.95

Member Price: €76.46

www.elektor.com/19948

Raspberry Pi Pico for Radio Amateurs + FREE Raspberry Pi Pico RP2040

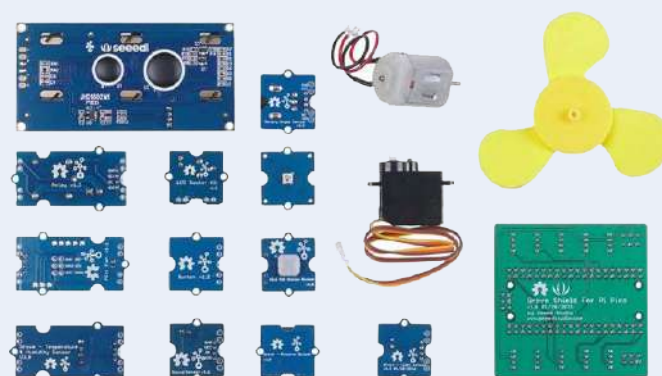


Price: €34.95

Member Price: €31.46

www.elektor.com/20041

Seeed Studio Grove Starter Kit for Raspberry Pi Pico



Price: €54.95

Member Price: €49.46

www.elektor.com/20016

HomeLab Tours

Everything Revolves Around the Tools...

By **Joachim Schröder** (Germany) and **Eric Bogers** (Elektor)

Anyone who is only even slightly serious about electronics will sooner or later not escape the need for some quality handiwork – read: wood and metal working – if only to give the circuit that was built with much ingenuity and effort a handsome or at least sturdy enclosure. And this requires a certain collection of tools.



Figure 1: An early prototype with a DC motor as the 'main spindle', a Nema 17 stepping motor for the lead screw and motor control using a simple PWM module.

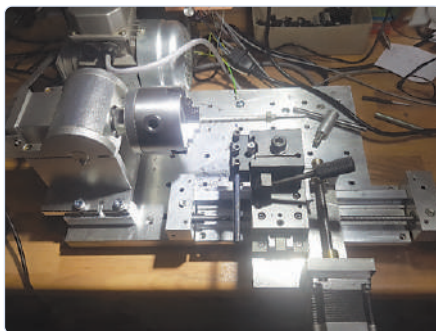


Figure 2: More comprehensive prototype with AC motor, variable speed controller and X and Z stepping motors.

If it only concerns a few holes, for the spindles of potentiometers or such, in a bought, ready-made enclosure, then we could (for starters at least) get quite a long way with just a jig saw (and a steady hand) plus a hobby drill with a not-too-wobbly drill stand. But the more involved mechatronics projects, which, for example need guide rods or similar, a (small) lathe quickly becomes indispensable — unless you are prepared to outsource all your mechanical work for relatively big money elsewhere.

Mr. Schröder has long been in possession of a manual operated lathe, which he (among other things) uses professionally for special assignments and repairs. He has, for some time, considered modifying this lathe into a CNC-lathe (CNC stands for *Computer Numerical Control*) with an electronically synchronised lead screw.

In such a synchronised lead screw, the carriage with the cutting tool is moved by a servo or stepping motor in synchronism with the rotation of the main spindle (for example 0.1 mm for each revolution); the main spindle has a rotary encoder fitted for this purpose. The advantage of such a CNC design is that when you need a larger or smaller displacement per revolution then there is no requirement to swap any gears.

In the end Mr. Schröder decided to build an entirely new CNC lathe — even if it was only because he couldn't afford to lose the use of the existing manual operated lathe. Initially, a few *proof of concept*-designs were built (Figures 1 and 2). These showed that one of the biggest problems is actually of a software nature: calculating when the stepping motor should advance one step — because of the

processing time it required, it was not possible to do this using floating-point calculations (which would have been the obvious approach). When the main spindle turns at (for example) 1200 revolutions per minute and an encoder is used which generates 1024 pulses per revolution, then this requires 20480 floating-point calculations per second. To convert these calculations to pure integer operations turned out to be one of the bigger challenges.

From China...

So a small (manual) lathe of Chinese provenance was ordered, that, in any case, has a sturdy cast frame (Figures 3 and 4). A good starting point, but those plastic gears are obviously completely inadequate (but they were going away anyway).

The rotary encoder for synchronising the lead screw is fitted with a gear that was salvaged from the original transmission and mounted with the aid of an adapter (Figures 5 and 6).

After only five days the original (Chinese) motor speed controller for the 230-V DC-motor gave up the ghost with a loud bang; it was replaced with an AC motor and a quality variable speed drive (Lenze). Figure 7 shows the definitive AC motor, rated 0.37 kW, including the variable speed drive.

Finally, Figure 8 shows the (provisional?) final result of all these efforts. Unfortunately, the brief scope of this article makes it impossible to go into the details of this project; anyone who is interested is invited to contact Mr. Schröder directly (see **textbox**).

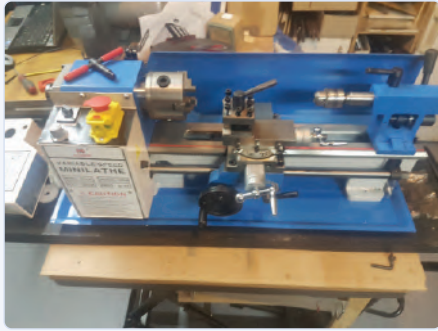


Figure 3: After a long wait the lathe arrived...

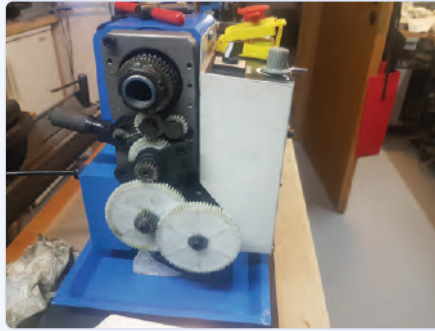


Figure 4: ...with a good chassis, but the gears (plastic) are clearly skimping on cost.

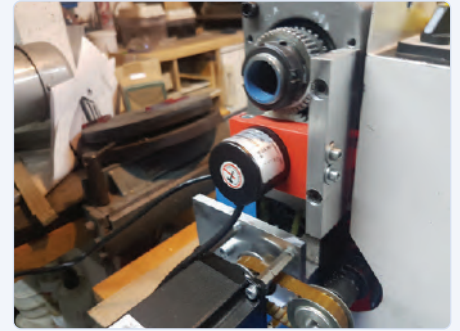


Figure 5: The encoder for synchronisation is mounted with the aid of an adapter...



Figure 6: ...and is driven via a gear salvaged from the dismantled gearbox.

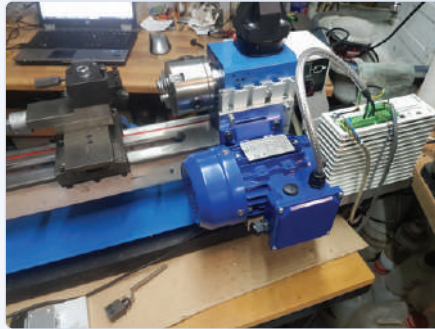


Figure 7: Here the 0.37-kW motor is mounted using an adapter plate.

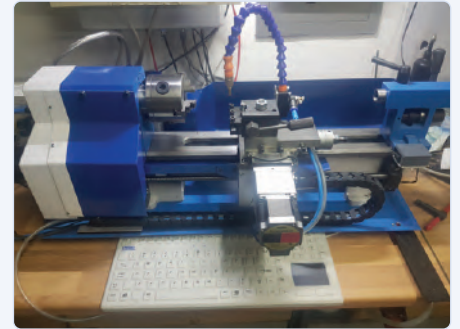


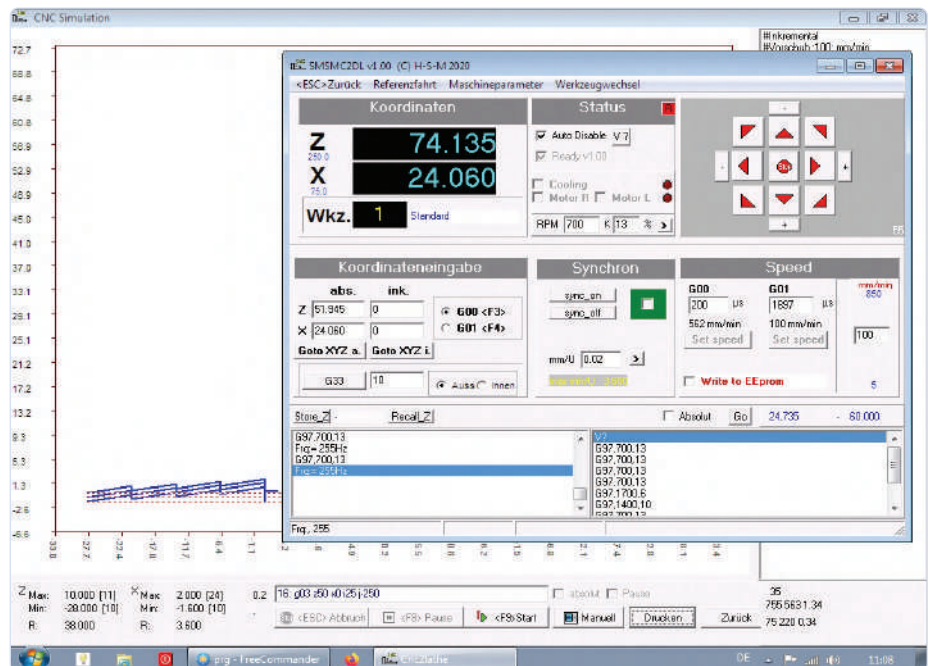
Figure 8: The result of all these efforts.

Figure 9: An impression of the software.

A Word About the Software

We briefly give the word to Mr. Schröder for the conclusion of this article: "More than 30 years ago, I built my first CNC milling machine and wrote the software for that myself. In those days that was under DOS, where the hardware was controlled directly via the LPT port. In the following years, I built newer and increasingly better machines, for which the software was ported to Windows. Because Windows does not offer real-time capabilities, I needed a microcontroller for controlling the axes. This problem I solved initially (around the start of this century) with the aid of Conrad CCBasic controllers; the absence of floating-point capabilities made this a real challenge however. Now I use Arduino boards and these work great – directly controlling 3D motion with speeds up to 1200 mm/min. It was obvious to use this software for the 3D CNC mill as the starting point for the 2D lathe. Easily said, easily done. **Figure 9** gives an impression of the resulting interface." ◀

210605-01



Contributors

Text and photos: Joachim Schröder
Editor: Eric Bogers
Translation: Arthur de Beun
Layout: Giel Dols

Questions or Comments?

Do you have any technical questions or comments prompted by this article? Send an e-mail to the author via js@hsm-ebis.de or to the editor of Elektor via editor@elektor.com.

Understanding the Neurons in Neural Networks (Part 4)

Embedded Neurons



By Stuart Cording (Elektor)

With our neural network fully functional on PCs and laptops, we can be confident in our multilayer perceptron implementation. Many applications require the low power capability and minimal latency that a microcontroller offers. Others simply do not want to share their private data with third-party cloud AI services. Here we make our neural network “Arduino ready” and move our traffic light classification code to the world of embedded systems.

Tools and platforms for machine learning (ML) seem to be sprouting up like mushrooms. No matter how complex your task or the size of your dataset, there is a cloud service that can handle it. However, there are many cases where a cloud-based ML implementation is unsuitable. If you are handling sensitive or personal data, you may not want to transfer it via the Internet to have it analyzed by an ML tool. Another example is automotive or other real-time embedded applications. Such systems require an immediate decision so the latency of an Internet connection, or the potential for having no connection at all, demands a local ML solution. This is also known as “ML at the edge” [1].

Operating a neural network at the edge means that even simple microcontrollers can execute ML algorithms. However, due to the processor demands required for training, the network is often trained in the cloud or on a powerful PC. The resultant weights are then downloaded to the microcontroller for Internet-free, low-latency operation.

This final article of the series ports our multilayer perceptron (MLP) neural network to an Arduino. Coupled with an RGB sensor, we teach it to classify the traffic light colors as we did in Processing on your PC or laptop. Using a 32-bit Arduino (Due or Mo Pro) we can undertake

the learning phase on the microcontroller. However, we will see that it is quite slow. Thus we will also partition the task so that the network is trained using a PC's performance, then executed in the low-latency, low-power environment of an embedded microcontroller application.

Selecting a Sensor

To recognize the traffic lights' colors, the Arduino will require a suitable sensor as an input. The TCS34725 is a suitable RGB sensing solution, available ready-to-use from Adafruit in a 2×2 cm ($0.8'' \times 0.8''$)

format PCB [2]. It allows us to acquire RGB data as we did previously with the camera and apply it to the same MLP configuration used in the previous PC-based examples. The board is suitable for use with both 5-V and 3.3-V Arduino's, and its sensor data is acquired using the I²C, which requires the Wire library.

To ensure consistent lighting of the sample to be analyzed, the board also includes a white LED controlled using an Arduino digital output. Pin 10 has been selected for this purpose (**Figure 1**). Thankfully, the

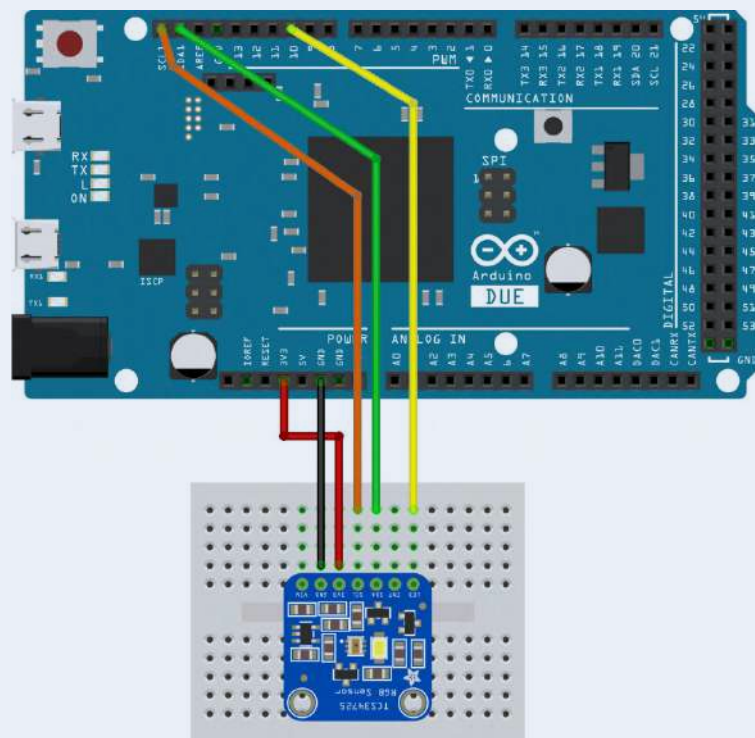


Figure 1: Wiring diagram for TCS34725 RGB sensor together with an Arduino Due.

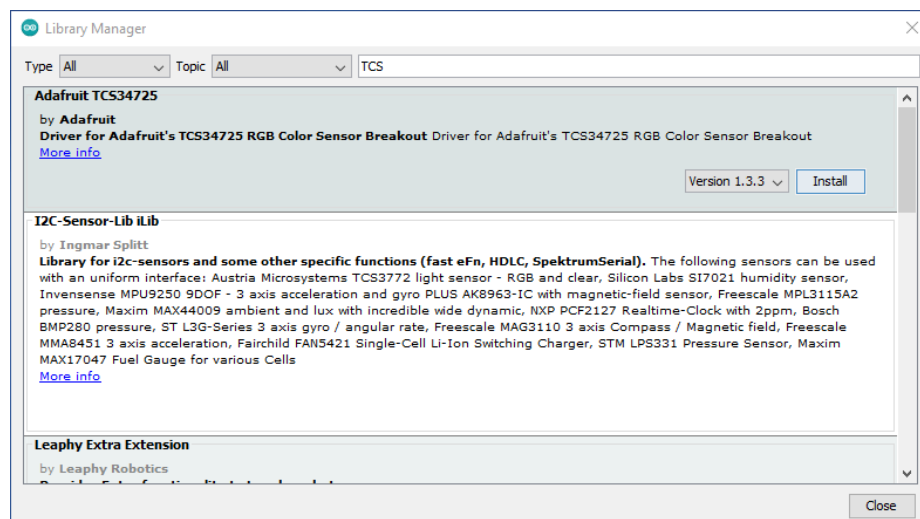
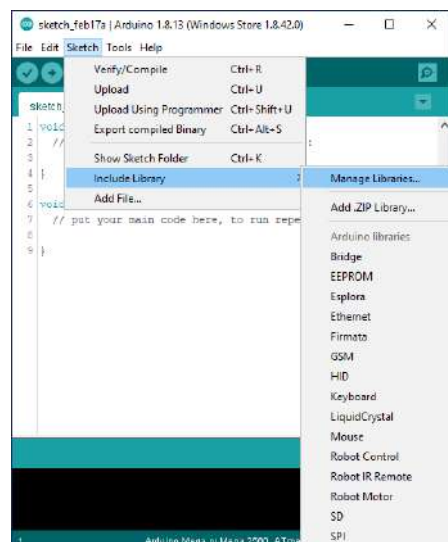


Figure 2: Installing the Adafruit TCS34725 RGB sensor software library in the Arduino IDE.

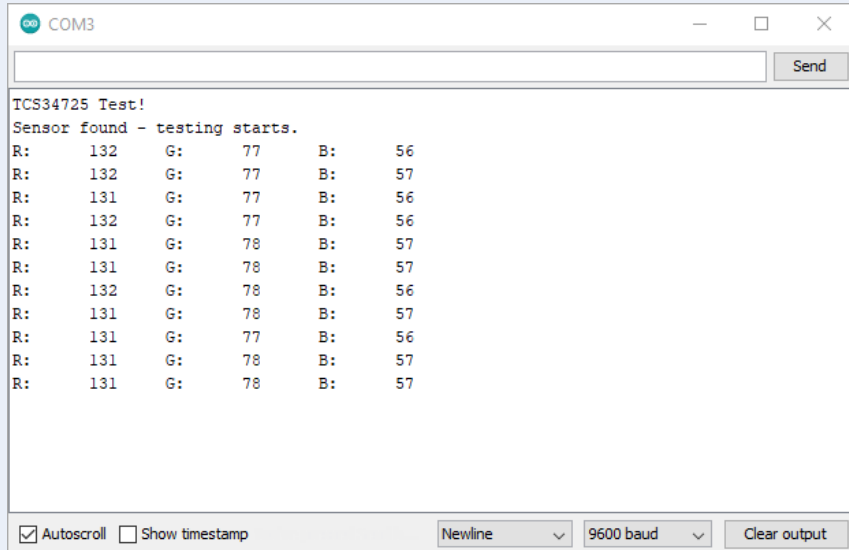


Figure 3: RGB value outputs provided by *tcsrgbsensor.ino* from Arduino.

board is supported by a well-constructed library, so getting up and running is simple and straightforward. As before, we will be using the code from the GitHub repository prepared for this series of articles [3].

Before running any code, we must install the library for the RGB sensor. Thankfully this should also be easy as it is accessible through the Library Manager within the Arduino IDE. From the menu bar, simply select *Sketch -> Include Library -> Manage Libraries...* and then enter “tcs” in the Library Manager’s search box. The Adafruit TCS34725 library should appear. Simply click *Install* to install it (Figure 2). Should there be difficulties, the source code and header file can be downloaded from the Adafruit GitHub repository [4] and added to the projects manually.

To ensure that the sensor works and provide a method to acquire the RGB values we need to train the ML, we will start with the sketch *arduino/tcsrgbsensor/tcsrgbsensor.ino*. After initializing the RGB sensor, the code turns on the LED and starts reporting the RGB values via the Serial output (Figure 3). Open *Tools -> Serial Monitor* to view them. The baud rate setting is 9600. To improve the quality of the readings and reduce the impact of other light sources, constructing a shield around the sensor board is worthwhile. A strip of black cardboard some 3 cm high and 10 cm long is ideal (Figure 4). The

shield also ensures that the traffic lights’ image can be held at a consistent distance from the RGB sensor.

With the RGB sensor delivering reliable results, we can now acquire its assessment of the red, amber, and green colors using our printed-out traffic light image (from [trafficLight/resources](#)). The results obtained by the author are shown in Table 1.

Table 1: RGB values captured using TCS34725 for the three traffic light colors.

Traffic Light	R	G	B
Red	149	56	61
Amber	123	77	61
Green	67	100	90

An MLP Library for Arduino

With the RGB values determined, we now need the neural network implementation for the Arduino platform. Because Processing uses Java, we cannot simply take the Neural class code and add it to an Arduino project. Thus the Java Neural class was modified slightly to turn it into a C++ class. In the Arduino world, such reusable code objects can be turned into ‘libraries.’ These consist of a C++ class and an additional file for highlighting the class’s keywords.

Should you wish to write your own library or better understand how C++ classes work on Arduino, there is an excellent tutorial on the Arduino website [5].

The code for our *Neural* library can be found in *arduino/neural*. The following Arduino projects simply include the *Neural* class source code in the sketch to keep things simple. The *neural.h* header file also needs to be added to the folder where the project is saved.

Using the *Neural* class on an Arduino is mostly the same as using it in Processing. Creating our *network* object as a global variable is slightly different and is undertaken as follows:

Neural network;

To construct the object with the desired number of input, hidden, and output nodes, we enter the following:

network = new Neural(3,6,4);

The basic configuration of the bias values and learning rate are then coded as previously in Processing:



Figure 4: TCS34725 RGB sensor with black card shield fitted.

```
network.setLearningRate(0.5);
network.setBiasInputToHidden(0.35);
network.setBiasHiddenToOutput(0.60);
```

From here, we can continue to use the methods as we used them previously in Processing.

Detecting Traffic Light Colors with Arduino

Now we can start exploring the MLP on the Arduino. The sketch `/arduino/tlight_detect/tlight_detect.ino` follows the same structure as the Processing project `tlight_detect.pde`. The neural network (3/6/4) is configured at line 40 onwards, and it is trained with the RGB data in a loop from line 51. Before executing the code, the RGB values for 'Red,' 'Amber,' and 'Green' acquired previously need to be entered at line 56 onwards:

```
teachRed(220, 56, 8);
teachAmber(216, 130, 11);
teachGreen(123, 150, 128);
```

Upload the code and open the Serial Monitor to view the output (**Figure 5**). As before, the baud setting should be 9600. The project checks that the RGB sensor is functional before turning off the white LED during learning. With the MLP configured, it then runs through the learning cycle 30,000 times, improving its ability to classify each of the three colors each time.

To provide some feedback during learning, a dot is output every 1,000 loop cycles (3,000 learning epochs). Compared to the PC, learning is a slow process. Each call to a learn function (`learnRed()`, etc.) requires around 5.55 ms to complete. Learning all three colors requires around 8.5 minutes on an Arduino Mo Pro using a SAMD21 microcontroller. If you wish to examine your platform's execution time, tick the *Show Timestamp* box in the Serial Monitor and replace line 66 with:

```
Serial.println(".");
```

This adds the carriage return character and

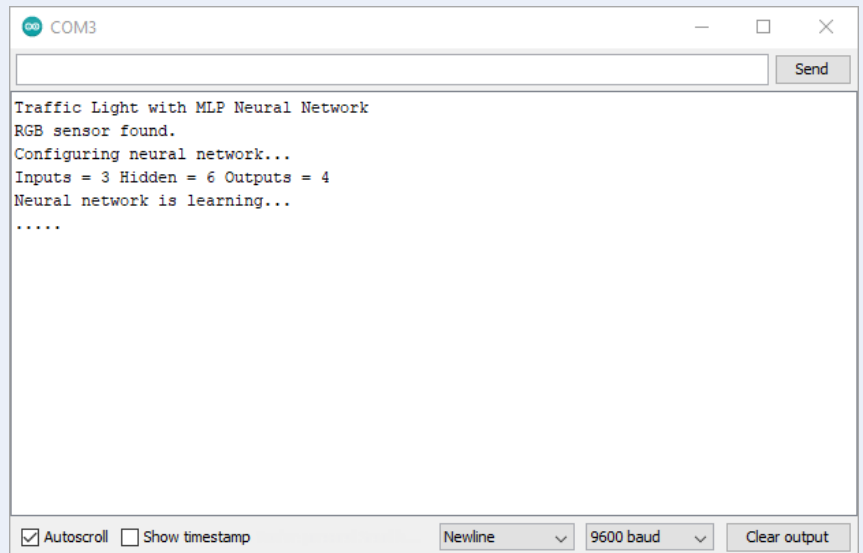


Figure 5: Output of `tlight_detect.ino` during learning.

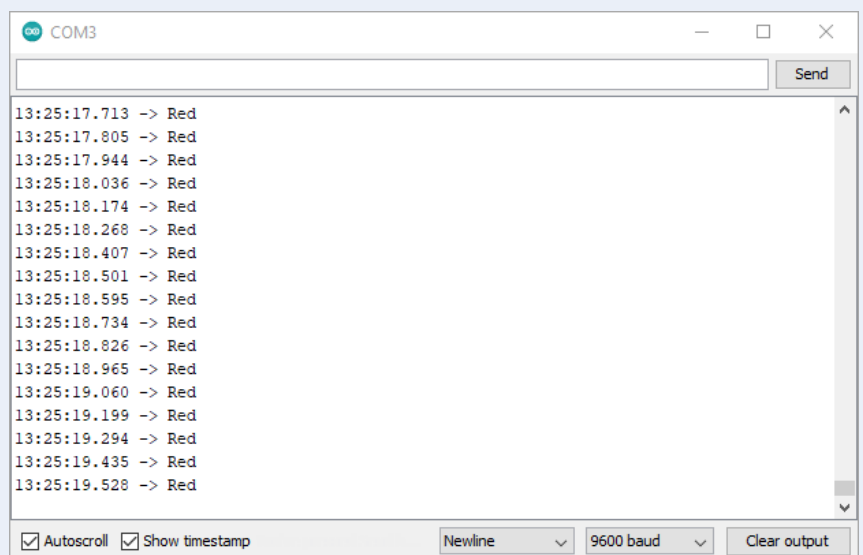


Figure 6: Output of `tlight_detect.ino` after learning and detecting colors.

ensures a timestamp is generated for each '.' character output.

Once learning is complete, the Arduino jumps immediately into showing off its new skill. Whenever a traffic light color is detected, it is output to the Serial Monitor (**Figure 6**). During the author's experimentation, the neural network also detected the color 'Amber' even when nothing was held in front of the sensor. Although this seemed to be related to ambient lighting, it highlights a weakness in the implementation.

To improve the code, we can teach the MLP 'Other' colors as we did previously in Processing. This can also be used to repress the 'Amber' classification when no image is present. The `tcsrgbsensor.ino` sketch can be used to acquire the sensor's readings for the traffic light surround, frame, and image background. These can then be input into lines 60 onward in the `tlight_detect.ino` sketch in the `teachOther()` function calls.

The values shown in **Table 2** were attained and tested with the `tlight_detect.ino` sketch.

```

Neural network is learning...
Input-to-hidden node weights
For Input Node 0: 0.9894259 -0.75018144 48.61366 -3.345678 9.416907 -23.454737
For Input Node 1: 2.1327987 5.392093 -36.850338 12.039759 -18.738537 15.558427
For Input Node 2: 1.3367374 -0.74704653 -1.242378 -5.9497995 -1.0344149 15.218534

Hidden-to-output node weights
For Hidden Node 0: -2.0546117 -1.203141 -2.7587035 -9.748096
For Hidden Node 1: -3.9066978 1.2856442 -0.48529842 -10.828738
For Hidden Node 2: 2.6418133 4.8958596 -25.040785 21.380386
For Hidden Node 3: -7.608626 6.0782804 4.0631976 -12.329156
For Hidden Node 4: 11.230279 -15.336227 -11.472162 -7.3535438
For Hidden Node 5: -14.677024 -18.876846 7.690963 20.697523

Arduino sketch code:

// For Input Node 0:
network.setInputToHiddenWeight( 0 , 0 , 0.9894259 );
network.setInputToHiddenWeight( 0 , 1 , -0.75018144 );
network.setInputToHiddenWeight( 0 , 2 , 48.61366 );
network.setInputToHiddenWeight( 0 , 3 , -3.345678 );
network.setInputToHiddenWeight( 0 , 4 , 9.416907 );
network.setInputToHiddenWeight( 0 , 5 , -23.454737 );

// For Input Node 1:
network.setInputToHiddenWeight( 1 , 0 , 2.1327987 );
network.setInputToHiddenWeight( 1 , 1 , 5.392093 );
network.setInputToHiddenWeight( 1 , 2 , -36.850338 );
network.setInputToHiddenWeight( 1 , 3 , 12.039759 );
network.setInputToHiddenWeight( 1 , 4 , -18.738537 );
network.setInputToHiddenWeight( 1 , 5 , 15.558427 );

// For Input Node 2:
network.setInputToHiddenWeight( 2 , 0 , 1.3367374 );
network.setInputToHiddenWeight( 2 , 1 , -0.74704653 );
network.setInputToHiddenWeight( 2 , 2 , -1.242378 );

```

Figure 7: Weights can be quickly calculated on a PC using *learnfast.pde* in Processing. The output of the text console can then be pasted into *tlight_weights.ino*.

The classification improved, but the false classification of no image as 'Amber' was not totally resolved. As always, there is room for improvement!

Table 2: RGB values captured using TCS34725 for the 'unwanted' colors.

Color	R	G	B
Dark traffic light surround	92	90	82
White frame	92	90	75
Blue background	73	93	89

Supercharged Learning

If you added learning the 'Other' colors to the Arduino sketch, you will have waited around 18 minutes for an Mo Pro to learn the desired classifications. This definitely leaves us some room for optimization of the process. Since we have a powerful PC that

can calculate the weights in under a second, it would make sense to do the learning there, then transfer the results to the Arduino. With these weights, we also have a means to program multiple microcontrollers with the same 'knowledge'. Providing the RGB sensors all function similarly with respect to our input, each microcontroller should correctly classify the traffic light image. So, this is what we will undertake next.

We return briefly to Processing to open the project */arduino/learnfast/learnfast.pde*. The entire application runs in the *setup()* function. The neural network is configured with the same input, hidden, and output nodes used on the Arduino board (3/6/4). In the learning loop (line 36), the values used are those acquired from the Arduino using the RGB sensor and the *tcsrcgsensor.ino* sketch. When the code is run, it outputs text to its console. The last section contains the code that configures all the input-to-hidden and hidden-to-output weights (Figure 7). Simply copy the code generated starting at *// For Input Node =0* until the end of the text output.

Back in the Arduino IDE we can now open the */arduino/tlight_weights/tlight_weights.ino* sketch. This is the same as the *tlight_detect.ino* sketch but, instead of teaching the neural network, the weights are preprogrammed. This happens at line 51 with the function *importWeights()*. Simply paste the code from the *learnfast.pde* output into the *importWeights()* function at line 86 in *tlight_weights.ino*. Program the Arduino board, and it should accurately detect the traffic light colors as before.


In fact, now that we have this supercharged learning process, we can even program the same *tlight_weights.ino* sketch into an Arduino Uno. Simply hook up the RGB sensor to the board, open the Serial Monitor, and it will function just as accurately as it did on an Arduino Mo Pro or Due. For comparison, you can monitor digital pin 9 to see how long it takes for the *calculateOutput()* method to perform the calculations.

What Next?

So, where do we go from here? Well, we have a functional MLP neural network that works on both a PC and a microcontroller. We also have a supercharged learning process to generate the weights needed in microcontroller applications. You can try and apply this MLP to other tasks that are too difficult to resolve using *if-else* statements and fixed limits. It may even be possible to implement a simple voice recognition project to recognize a few words. You could also explore the following:

- The Neural class uses the data type *double*. Can it be sped up using *float* and still maintain accuracy? How much faster can it run?
- The sigmoid function uses the *exp()* math function, calculating the exponential raised to the argument passed. Could the activation function be simplified using an approximation and still deliver accurate classification?

- If you wanted to attempt voice recognition, how would you prepare a voice sample to present it to this MLP implementation?
- How about some significant changes? How would you implement a second layer of hidden nodes? Can you implement a different activation function?

In this series of articles, we've covered a lot of ground. We scratched the surface of the early artificial neuron research. From there, we examined how MLPs use backpropagation to learn and investigated its operation using the powerful processing available on a PC. We then ported the MLP to a microcontroller as an example of edge ML. Should you choose to develop these examples further, feel free to share your results with us here at Elektor. 

210160-D-01

Questions or Comments?

Do you have questions or comments regarding this article? Then email the author at stuart.cording@elektor.com.

Contributors

Idea, Text and Images: **Stuart Cording**
 Editors: **Jens Nickel** and **C. J. Abate**
 Illustrations: **Patrick Wielders**
 Layout: **Giel Dols**

Better Farming

Farming has always relied on nature and the seasons to determine the best time to reap and sow. Tradition had always dictated the optimal date to plant their crop to benefit from the monsoon season. However, with changing weather patterns, crop yields have suffered. All this is changing thanks to AI. Indian farmers participating in a pilot project planted their groundnut crop in late June, three weeks later than normal. The Microsoft Cortana Intelligence Suite provided this guidance. Using historical climate data, the recommendations the farmers received led to a 30% average higher yield [6].



 **RELATED PRODUCTS**

- B. van Dam, Artificial Intelligence (E-book) (SKU 18090)
www.elektor.com/18090
- Google AIY Vision Kit for Raspberry Pi (SKU 19365)
www.elektor.com/19365
- HuskyLens AI Camera with Silicone Case (SKU 19248)
www.elektor.com/19248

WEB LINKS

- [1] M. Patrick, "ML at the Edge: a Practical Example," codemotion, September 2020: <https://bit.ly/2ZQYipU>
- [2] RGB Color sensor with IR filter and white LED - TCS34725: <http://bit.ly/2NKFS7T>
- [3] GitHub repository - simple-neural-network: <https://bit.ly/2ZHLv9p>
- [4] GitHub repository - Adafruit_TCS34725: <http://bit.ly/37RJg81>
- [5] "Writing a Library for Arduino," Arduino: <http://bit.ly/3aWeNaP>
- [6] "Digital Agriculture: Farmers in India are using AI to increase crop yields," Microsoft News Center, November 2017: <http://bit.ly/2PacsQZ>

Magnetic Levitation

the Very Easy Way

The Third and Most Compact Version



By Peter Neufeld (Germany)

Let's take a look at a simple analog circuit for levitating small objects, which is the subject of many interesting discussions and experiments. The hardware is similar to the design we already published in *Elektor*, but with even fewer components. The electronics now even fit in the enclosure of a modified industry-standard relay!

I have written on the Elektor Labs platform about my experiments with magnetic levitation circuits. The first two versions were published online and in *Elektor Magazine*: one was a completely analog solution with an LM311 comparator [1], and the second proposed a digital solution [2], controlled by an ESP32-based microcontroller module. Here, I will refer to these articles as Part I and Part II, respectively. During some helpful discussions with Elektor Lab engineer Luc Lemmens about magnetic levitation, some thoughts came up which I followed up to improve and optimize the projects I had already made. Some of these changes were already implemented in the circuits in these two publications, but the discussions also triggered thoughts on further optimization. The result was another strong simplification of the "old" analog circuit that allows me to levitate small (magnetic) objects, which we are going to discuss in this article. This is the third and — at least for the time being — last part about my magnetic levitation projects. Even though the design is simpler compared to the other two projects, the functionality is very good.



The Principle, in Short

The idea behind all three magnetic levitation projects is that only a small part of the force required to levitate an object comes from an electromagnetic field of a modified relay coil. The actual heavy work is done by the permanent force between a permanent magnet in the levitated object and the iron core of the solenoid. A Hall sensor measures the strength of the total magnetic field (i.e., the sum of the static and electromagnetic field). The voltage level on its output pin is also a measure for the distance between the coil and the levitated object, and it is used as a feedback signal to control the current through the electromagnet, which — in turn — controls the distance between the core and the levitated object.

A preset potentiometer is used to fine-tune the control circuit and to adjust it to the magnetic object that should hover below the coil. When you start experimenting with these circuits, the adjustment may prove to be a real challenge. If you can't get it to work, the first article [1] can provide useful tips and tricks. It can be done, but of course there are limits to the size and weight of object that can be lifted with the limited power of this circuit.

Simplified Circuitry

The setup for the version presented in this article is shown in **Figure 1**, which is a stripped-down version of the first installment. Here, an LM311 comparator [3] provides the relatively small control current through the coil without the use of an additional external driver transistor. As you may know, the LM311 has an open-collector output, which can drive a small load. This simplifies the circuit considerably compared to the first completely analog solution I designed for Part I, where a BC550 NPN-transistor was added to the output of the comparator to power the electromagnet.

To reduce the number of components even further and to save some more space, the adjustable voltage divider for the reference level on the comparator's inverting input (P1 and R4) can also be modified. These two components can be replaced with a single multiturn trimmer; its value isn't critical, anything in the range of 1 k Ω to 1 M Ω will do. As you can see in **Figure 2**, the single-turn preset potentiometer plus fixed resistor R4 will fit into the relay's enclosure too, so it isn't really necessary to do this mod.

The components used here are very common, readily available and affordable. In Part I, we wrote that cheaper and more readily available Hall sensors like the SS49E were *not* useable and that only the Allegro Microsystems A1302 or A1308 would do the trick. However, in Part II — levitation controlled with a microcontroller — we were happy to report that this assumption was incorrect. Further tests proved that the type and brand of IC2 really isn't that critical, and this makes it a lot easier to find the components needed to build these projects. The Allegro sensors are more expensive and difficult to find.

How to Get the Right Relay and Modify It

The electromagnet (L1) now consists of the solenoid of an industry-standard 5 V relay, a Finder relay with type number 40.52.7005 to be more precise, which is readily available and considerably easier

to modify than the 5 V relays we used before. The complete process of modifying the relay is illustrated by the photos in **Figure 3**.

The transparent housing of a Finder relay is often glued to its base. However, it can be easily opened by heating the adhesive joint with a hair dryer, or hot air soldering station at moderate temperature. An Exacto knife, also slightly heated, can be useful to open the enclosure

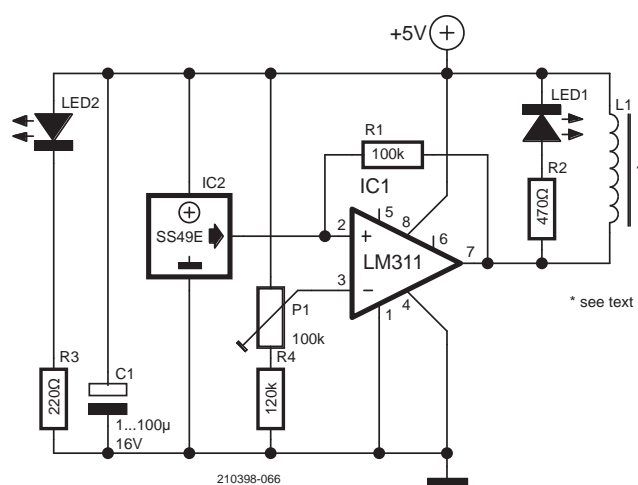


Figure 1: The schematic diagram of this version of the magnetic levitation.



Figure 2: A single turn trimmer plus series resistor also fits into the enclosure.



Figure 3: Taking the relay apart and preparing the solenoid.

without doing any damage and that is exactly what we want here: to keep the cap intact to be reused as an enclosure for the complete electronics of this levitation device.

Just like with the relays used in the earlier publications, the core of the solenoid must be changed from its U-shape to an I-shape to avoid a magnetic short. This is quite easy because the round core and the rectangular flat extension are only peened together. This joint can be removed with a small cutting disc, a milling head or a drill.

Finally, the iron core can be taken out — if it doesn't fall out by itself — and turned to obtain a slightly larger and also flat pole surface to glue the HALL sensor on, as shown in **Figure 4**. At least the electromagnet and the Hall sensor can be mounted within the relay's housing, where they are well protected against the magnet.

The current consumption with a magnetic object hovering under the coil is only 50 mA at an operating voltage of 5 V, and up to a maximum of 90 mA when the coil is permanently powered (e.g., when there is no magnet near the core).

The modification of the Finder relay creates a very good electro-magnet and — with the cap — even a good mechanical protection of the whole circuit against permanent magnets hitting the core and/or Hall sensor. If you really want to build it that small within the former relay housing — like I did — it can be done with acceptable mechanical effort. It will take some time to figure out how the components can be arranged in this relatively small volume. But, of course, the circuit can also very easily be built on a breadboard or grid board (see **Figure 5**), and only four wires are needed to connect

the combination of coil, LED, resistor and Hall sensor to the rest of the circuit. A 20 cm long piece of flat band cable was very helpful to me for this purpose in a prototype.

Good Engineer or Brave but Reckless Tinkerer?

Luc from the Elektor Lab Team has critically, but absolutely correctly, analyzed that the LM311 can sink 50 mA at its output. But what about 90 mA? Well, this remark is correct. But my goal was to keep the circuit as simple as possible and to reduce the number of components to an absolute minimum. And so, the tinkerer in me didn't want to just use an additional driver transistor to prevent my circuit from dying prematurely.

A closer look at the internal circuitry of the LM311 (**Figure 6**) and diagrams in the datasheet revealed that the chip has an internal protection circuit that limits the output current to a harmless level, even in case of an output short circuit and ensures a maximum power dissipation of 350 mW at 5 V supply voltage. This makes the output almost immortal — or at least insensitive to a low impedance load. Finally, I kept my circuit switched on for two days, and as I expected and hoped, it survived!

OK, let's take a step back. I respect speed limits, I pay my taxes, and I read and follow datasheets — and I know many very good reasons to stay within the safe operating specs of semiconductors. Even if integrated circuits have all kinds of protection circuits, it is bad practice to push them to the limits. It will most definitely shorten the lifespan of the chips. You may argue that — in this case — it will still last long enough and that replacing an LM311 doesn't cost a fortune, but with this compact construction it will be a pain to repair the circuit.

Better safe than sorry, but it was a challenge for me to try out if the circuit can work with lower output current. An easy and obvious way to achieve this is to replace the 5 V Finder 40.52.7.005.0000 relay with 50 Ω coil resistance by a 6 V relay with 75 Ω (Finder 40.52.7.006.0000), but I did not have such a relay on hand.

So, I turned to the next best solution: while keeping the 5 V supply, I reduced the voltage across the coil by inserting two 1N4148-diodes in series between the power supply and the solenoid (see **Figure 7**). This reduced the coil current with no payload to 50 mA and with payload to an average of 39 mA. A single 22 Ω resistor will have the same effect. And that works well too! This will keep the comparator within its electrical specs. Unfortunately, this power reduction also reduces the distance between the electromagnet and the hovering object. What is worse, the control system has less margin to absorb disturbances. In other words, at this lower power level, the floating object will fall down or hit the electromagnet even if it deviates just a little from the adjusted distance to the core. The circuit is less able to correct it.



Figure 4: The Hall sensor glued to the iron core of the coil.

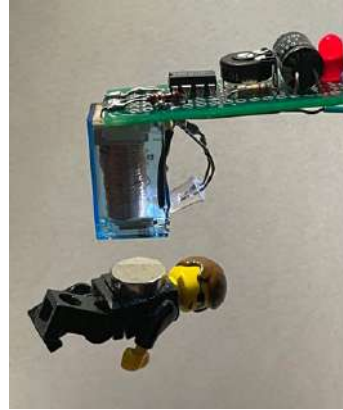


Figure 5: This circuit can also be constructed on a piece of Veroboard.

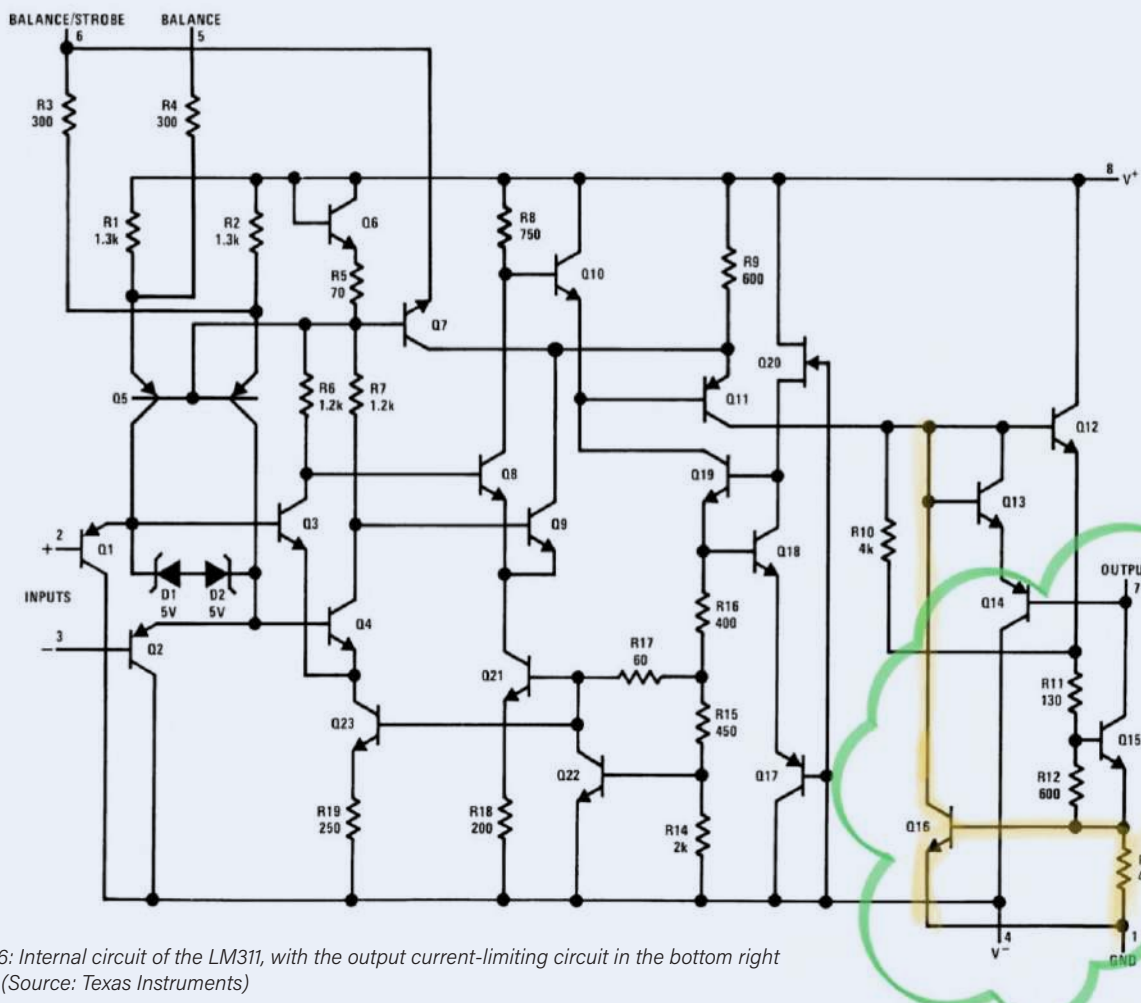


Figure 6: Internal circuit of the LM311, with the output current-limiting circuit in the bottom right corner. (Source: Texas Instruments)

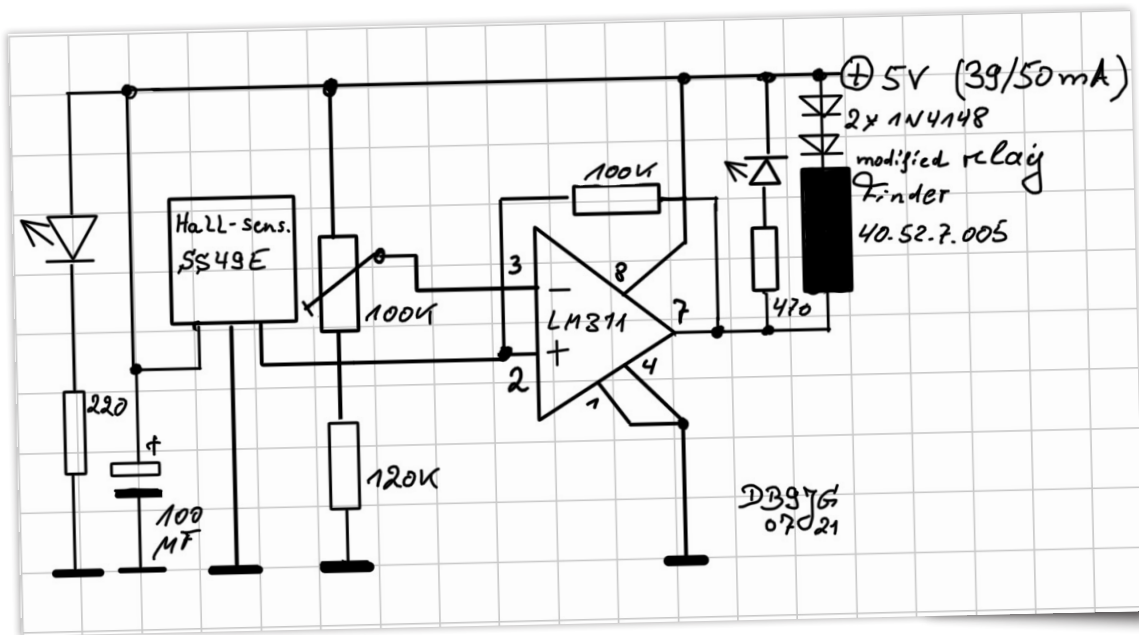


Figure 7: The magnetic levitation circuit with two 1N4148 diodes to limit the current through the coil.

Design Ideas?

All in all, I like this smaller installment more than the first analog version I built on Veroboard for Part I. It looks nicer, and the electronics are better protected by the enclosure of the relay. The second version — using an ESP32-based M5Stack Atom — showed that the analog comparator could be replaced with a microcontroller and a simple algorithm. In my opinion, the different variants of this small project, with its actually very general control engineering problem, showed that an analogue design still has its advantages, even in today's very microcontroller-dominated times!

As I announced at the beginning of this article, this is the last and final version of my magnetic levitation projects, unless... Maybe you have ideas or suggestions how to improve the design even further?

210398-01

Questions or Comments?

Do you have technical questions or comments about this article? Feel free to contact the editor at luc.lemmens@elektor.com!

Contributors

Design and text: **Peter Neufeld**

Editing: **Luc Lemmens**

Illustrations: **Peter Neufeld, Patrick Wielders, Luc Lemmens**

Layout: **Harmen Heida**



COMPONENT LIST

Resistors

R1 = 100 k
R2 = 470 Ω
R3 = 220 Ω
R4 = 120 k
P1 = 100 k trimmer

Capacitors

C1 = 1 μ F...100 μ F, 16 V radial

Semiconductors

LED1, LED2 = green LED
IC1 = Hall sensor SS49E
IC2 = LM311 comparator

Miscellaneous:

L1 = 5V relay, Finder
40.52.7.005.0000 (see text)



RELATED PRODUCTS

> **Velleman Helping Hand with Magnifier, LED Light and Soldering Stand (SKU 19864)**
www.elektor.com/19864

> **Velleman VTSS220 Temperature-Controlled Soldering Station (SKU 19865)**
www.elektor.com/19865

WEB LINKS

[1] "Magnetic Levitation the Easy Way," ElektorMag 7-8/2021: <http://www.elektormagazine.com/200311-01>

[2] "Magnetic Levitation the Digital Way," ElektorMag 9-10/2021: <https://www.elektormagazine.com/200278-01>

[3] LM311 comparator datasheet: <https://www.ti.com/lit/ds/symlink/lm311-n.pdf>

PLC Programming with the Raspberry Pi and the OpenPLC Project

Visualization of PLC Programs with AdvancedHMI

By **Josef Bernhardt** (Germany)

In industrial control systems, as well as in home automation, it's accepted practice to control and observe the current process using so-called monitors. In this chapter extracted from Josef Bernhardt's latest book on PLC programming, you will get acquainted with the AdvancedHMI software. You access the variables in the PLC and try to visualize them. It is also possible to access the process and simulate switches with it.

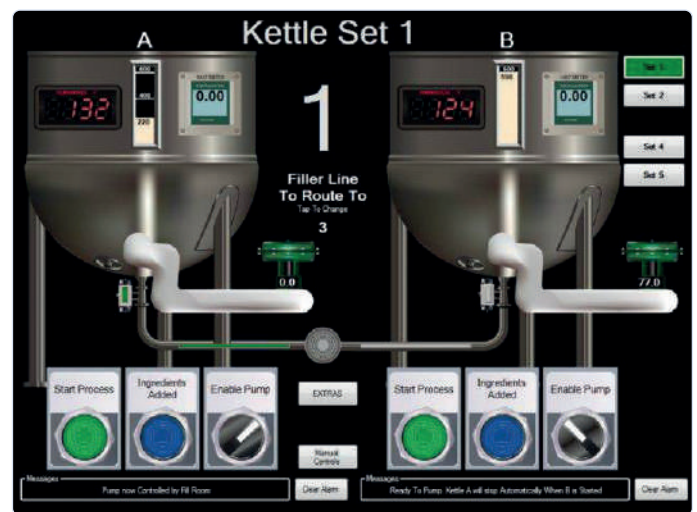


Figure 1: AdvancedHMI example.

AdvancedHMI is a .NET program written in Visual Basic using Microsoft's Visual Studio Community software. If you install the Mono Framework on your Raspberry Pi, you can also run the software on it and get access to the running PLC program. Both the AdvancedHMI and the Visual Studio Community software can be downloaded for free. An example of AdvancedHMI in action is pictured in **Figure 1**.

First, download the software from the manufacturer's website and write a test program for your Raspberry Pi PLC. To download it, go to the manufacturer's website, www.advancedhmi.com, and add the "AdvancedHMI Base Package" to your shopping cart, as shown in **Figure 2**. After that, click on the *Checkout* box.

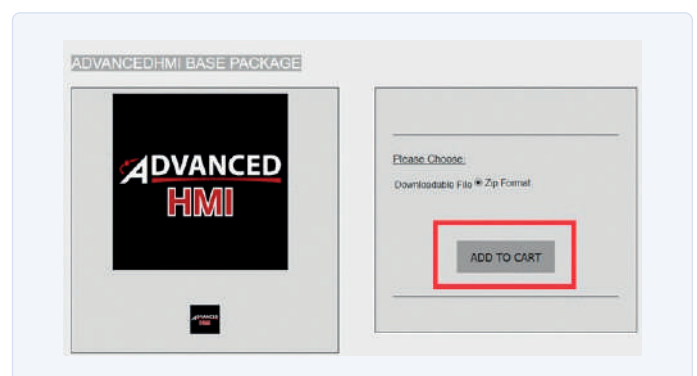


Figure 2: Obtaining AdvancedHMI.

Editor's Note. This article is an excerpt from the 194-page book *PLC Programming and the OpenPLC Project — ModbusRTU and ModbusTCP examples using the Arduino Uno and the ESP8266*. The excerpt was formatted and lightly edited to match Elektor Mag's editorial standards and page layout. Being an extract from a larger publication, some terms in this article may refer to discussions elsewhere in the book. The Author and Editor have done their best to preclude such instances and are happy to help with queries. Contact details are in the [Questions or Comments?](#) box

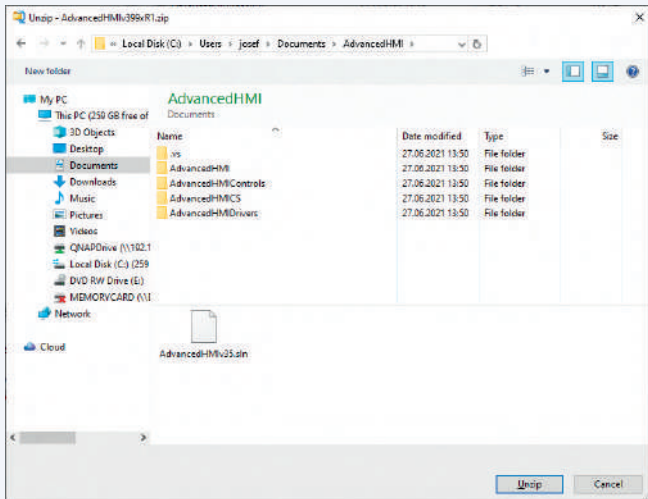


Figure 3: AdvancedHMI unzip action.

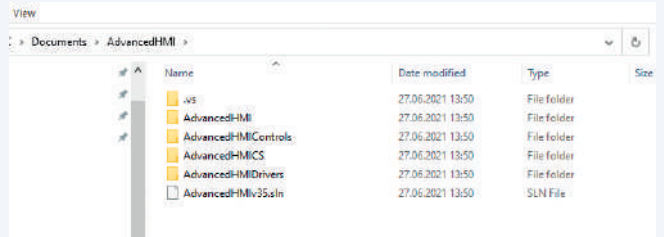


Figure 4: AdvancedHMI project directory.

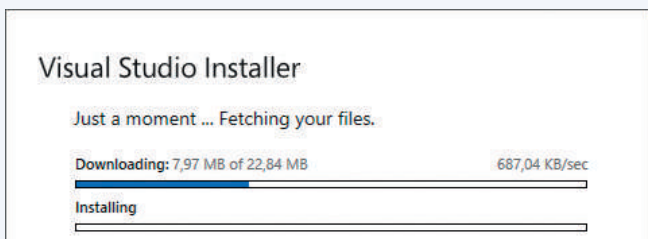


Figure 5: Visual Studio Installer.

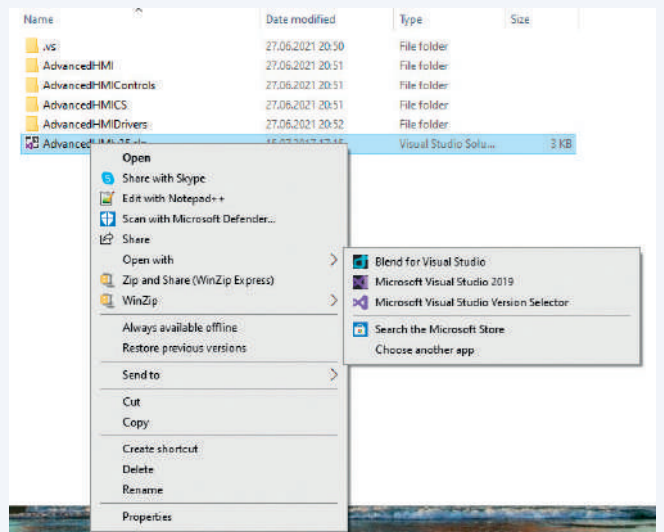


Figure 6: Open your AdvancedHMI project.

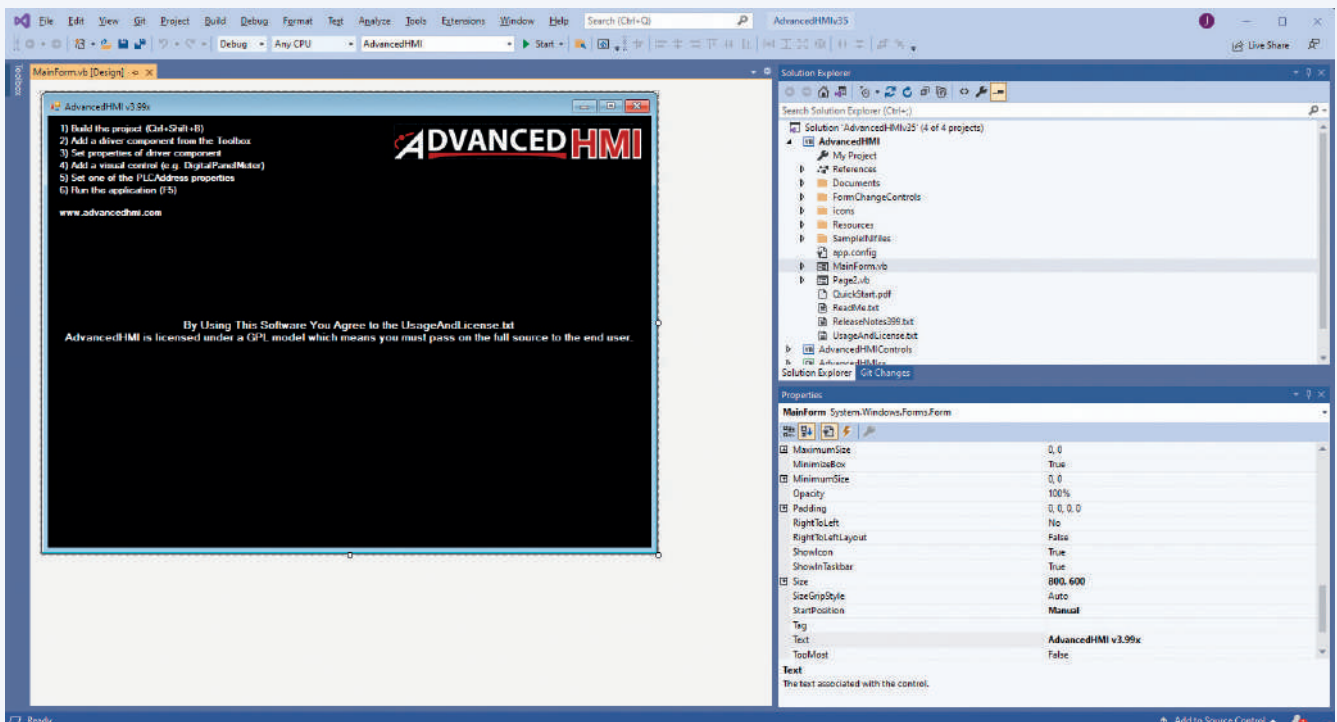


Figure 7: Visual Studio user interface.

Preface to the Book

This book is intended to provide readers with a practical introduction to using the Raspberry Pi computer as a PLC (programmable logic control) for their projects.

The project is indebted to programmers Edouard Tisserant and Mario de Sousa. They started the "Matic project" after the introduction of IEC standard 61131-3 in 2003. This made it feasible to translate the programming languages introduced in the standard into C programs.

Later, when the Raspberry Pi became increasingly popular, Thiago Alves started the "openplcproject". He extended the editor from the "Beremiz" project and wrote a runtime library and a web interface for the Raspberry Pi and the PC. From then on, it was possible to write programs on the PC and install them on the Raspberry Pi.

Many Raspberry Pi users are now able to realize their own controls and regulation systems using their own hardware. The hardware and software are also excellent for training purposes because it abides by the IEC standard.

Beginners will also learn everything about installation and programming in the five programming languages in order to build their own control systems. In a later chapter, the visualization with AdvancedHMI is discussed to display processes on the screen.

Circuits with the Arduino and ESP8266, which are necessary for Modbus, are also explained.

After the registration procedure, the program can be downloaded for free and unpacked into a directory (**Figure 3**). Save the ZIP file as a template.

Next, create a folder *AdvancedHMI* and copy the content of the file *AdvancedHMIv399xR1.ZIP* into this directory (**Figure 4**). To open the project, a development environment is needed, so proceed to install the *Visual Studio Community* software from the Microsoft website, <https://visualstudio.microsoft.com/vs/community/>, as shown in **Figure 5**.

After the installation process is complete, let us start the first demo project. With a right mouse click, we open the *AdvancedHMI* solution with Microsoft Visual Studio 2019 (**Figures 6 and 7**). With a double-click on *MainForm.vb*, you will get the still vacant HMI view of your demo project (WinForm). Now you can run the program by clicking on *Start* (**Figure 8**). At this point, the project has been compiled and saved in the directory shown in **Figure 9**.

These project files are needed if you want to use a program on another computer. You can also run these files on the Raspberry Pi with the Mono Framework.

Now, drag and drop the component *ModbusTCPCom* from the left toolbox (**Figure 10**) onto your worksheet. In the Properties window, you can now enter the IP address of your Raspberry Pi (**Figure 11**).

To test the communication, create a new PLC program with the Open PLC Editor (**Figure 12**), load it to the Raspberry Pi, and start it. This is a simple On/Off circuit using the *On* and *Off* buttons (**Figure 13**).

It gets interesting with the button *BTN_HMI*, which is at address %QX2.0. This address — outside the address range of the Raspberry

Pi's GPIOs — is necessary for the HMI interface to trigger a button press. With the timers *TONo* and *TOFo*, you can build a blinking LED circuit for the *LEDBlink* program.

Now switch back to your HMI project and drag a switch (*MomentaryButton*) and three indicator lights (*PilotLight*) from the toolbox, and label them as you can see in the end result, **Figure 14**. To do this, select the object and change the text in the Properties window to those shown.

Set the fields *PLCAdressClick* as follows:

- **BTN_HMI** is at address 17, this is because you used %QX2.0 for the *BTN_HMI* (8+8+1).
- **LED** goes to: 1
- **LEDModbus** goes to: 2
- **LEDBlink** goes to: 3

The *MomentaryButton* can be changed from button to switch under the properties of *OutputType* as shown in **Figure 15**. With this feature set, you have the correct addresses to have access to the variables of your PLC program. Now you can start your program and run a test.

The three LEDs always show the same state as the LEDs from the test board. The switch *BTN_HMI* allows you to switch LED 2 on the test board.

Table 1 shows the Modbus address mapping for the Raspberry Pi with OpenPLCproject and AdvancedHMI. OpenPLC also provides a separate address space for memory variables with support for 16-, 32-, and 64-bit variables. This is summarized in **Table 2**.

For the installation of the Mono Framework on the Raspberry Pi,

Function Code	Usage	PLC Address	Modbus Register	Register Size	Value Range	Access	Advanced HMI	Example
FC01 Read Coil	Digital Outputs	%QX0.0 - %QX99.7	0-799 (1-800)	1 Bit	0 or 1	Read / Write	"00001" - "00800"	LED
FC02 Discrete Input	Digital Inputs	%IX0.0 - %IX99.7	0-799 (1-800)	1 Bit	0 or 1	Read Only	"100001" - "100800"	BTN
FC04 Input Register	Analog Inputs	%IW0 - %IW99	0-1023 (1-1024)	16 Bit	0-65535	Read Only	"30001" - "31024"	ADC
FC03 Holding Register	Analog Outputs	%QW0 - %QW99	0-1023 (1-1024)	16 Bit	0-65535	Read / Write	"40001" - "41024"	DAC

Table 1

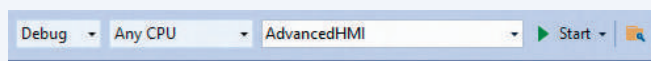


Figure 8: Visual Studio menu bar.

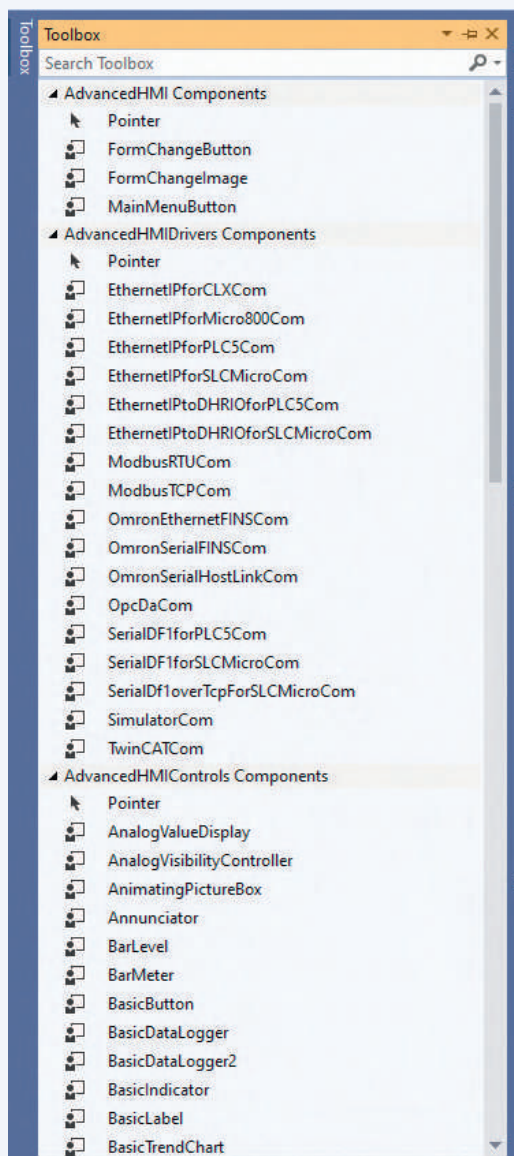


Figure 10: Visual Studio Toolbox.

#	Name	Class	Type	Location	Initial Value	Option	Docum
1	BTN_ON	Local	BOOL	%IX0.0			
2	BTN_OFF	Local	BOOL	%IX0.1			
3	LED	Local	BOOL	%QX0.0			
4	LED_Modbus	Local	BOOL	%QX0.1			
5	LED_Blink	Local	BOOL	%QX0.2			
6	BTN_HMI	Local	BOOL	%IX2.0			
7	TON0	Local	TON				
8	TOF0	Local	TOF				

Figure 12: PLC example variable list.

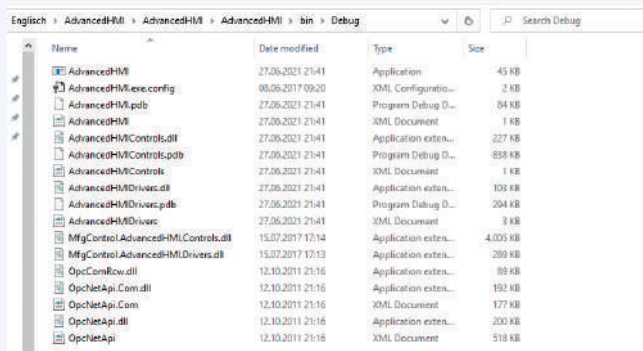


Figure 9: AdvancedHMI debug directory.

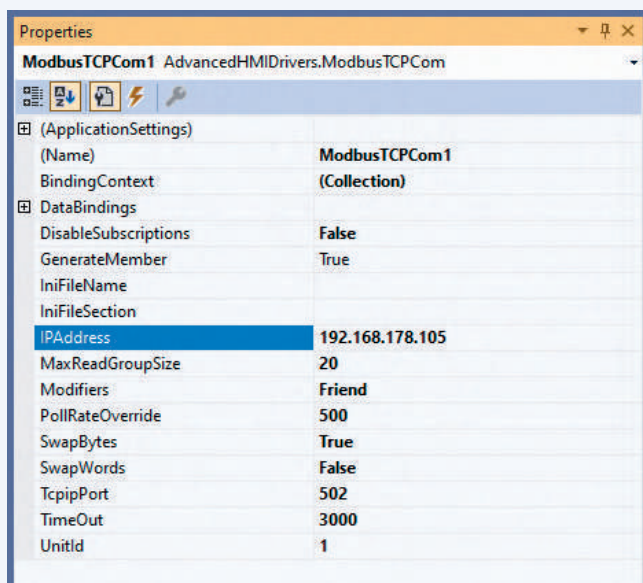


Figure 11: Modbus settings (IP and port).

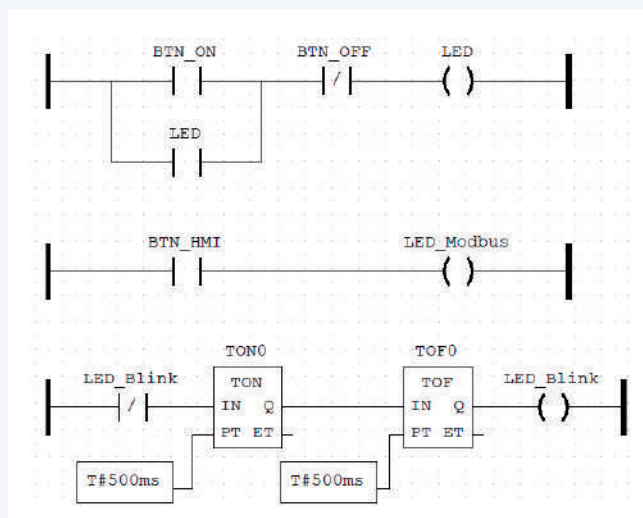


Figure 13: LED sample program.

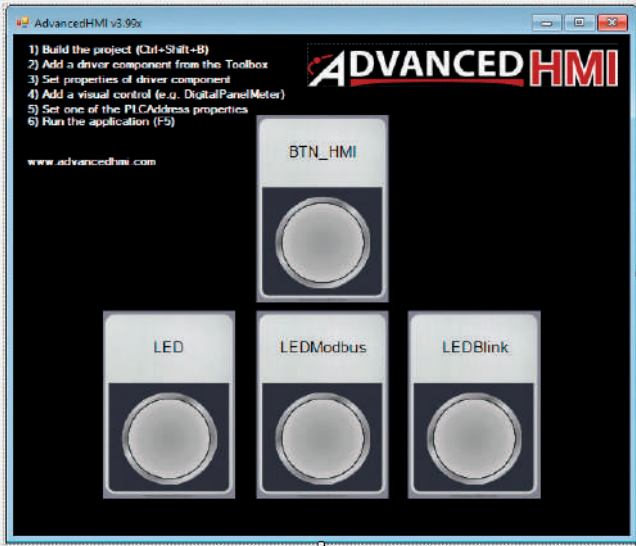


Figure 14:
HMI program.

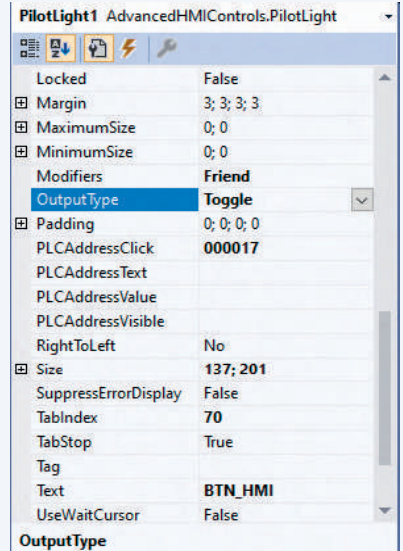


Figure 15: HMI address
setting.

the following commands are needed:

```
sudo su
apt-get update
apt-get install mono-complete
apt-get install mono-vbnc
```

Now you can create a directory on the Raspberry Pi and use WinSCP to copy your project from the PC to the Raspberry Pi and test it.

The example program can also be found in the download directory under *AdvancedHMI*. The .EXE files for the examples discussed in the book are stored in the following subdirectory:

`PLC-Book-Download\AdvancedHMI\AdvancedHMI\bin\Debug`

The subdirectory is contained in the software bundle released by the author in support of his book. The software is available for free downloading. Head over to [1], scroll down to the Downloads, and click on the file name:

Software PLC Programming with the Raspberry Pi and the OpenPLC Project. Save the ZIP archive file locally (128.94 MB) and then extract it. ◀

210642-01

Questions or Comments?

Do you have any technical questions or comments related to this article? Email the author at josef@bernhardt.de or Elektor at editor@elektor.com.

Contributors

Text: Josef Bernhardt

Editor: Jan Buiting

Layout: Giel Dols



RELATED PRODUCTS

- **Book:** J. Bernhard, *PLC Programming with the Raspberry Pi and the OpenPLC Project* (SKU 19966)
www.elektor.com/19966
- **E-Book:** J. Bernhard, *PLC Programming with the Raspberry Pi and the OpenPLC Project* (SKU 19967)
www.elektor.com/19967

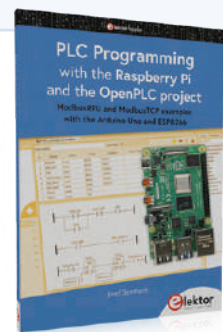


Table 2

Register Type	Usage	PLC Address Range	Modbus Address	Register Size	Value Range	Access	Advanced HMI
Holding Register	General 16Bit Register	%MW0 - %MW1023	1024..2047 (1025-2048)	16 Bit	0-65535	Read / Write	"41025" - "42048"
Holding Register	General 32Bit Register	%MD0 - %MD1023	2048..4095 (2049-4096)	32 Bit	0-4.294.967.295	Read / Write	"42049" - "44096"
Holding Register	General 64Bit Register	%ML0 - %ML1023	4096..8191 (4097-8192)	64 Bit	0-huge	Read / Write	"44097" - "48192"

WEB LINKS

[1] Book resources/info page: <https://www.elektor.com/plc-programming-with-the-raspberry-pi-and-the-openplc-project>

From Life's Experience Pack Up and Leave

By Ilse Joostens (Belgium)

We live in a big bad world, and normally not a day will go by without becoming annoyed at some of the most insignificant things. On the long list of small and large annoyances that can drive us up the wall, packaging features prominently. Take the blister pack as an example. These blasted things have to be attacked with scissors, knives and other sharp tools in order to open them. With a bit of bad luck you will not only cut your hand on the sharp plastic, but with all that violence, whether accompanied by some strong language or not, you also cut or damaged the product. Or what do you think about packaging that is easily unpacked, but where it is absolutely impossible to put everything back in the box, should that become necessary? Returns are fairly common, after all. Welcome to the wonderful world of packaging!

Inverse Tetrismatics

In the article about the sand clock, published in January 2017 [1], I described the principle of "inverse kinematics", of which I will spare you the details in this column. If desired, as a form of self-punishment, you are welcome to read the relevant article. Around that time I stood with my better half in a queue at the checkout in the local supermarket. Here we made a sport out of arranging the articles from the shopping cart in such an order on

the checkout conveyor, that after scanning we could put them quickly in our bags, with the heaviest items at the bottom. Because we do enjoy the odd word game every now and then, the term "inverse tetrismatics" was born.

As an electronics enthusiast, your attention is naturally focused on the electronics side of things, such as: schematics, components, circuits, circuit boards, modules, measuring equipment and much more. The mechanical

aspects, such as enclosures and front panels are generally less beloved and for many a cold sweat will break out at the mere thought of it. Standing for hours filing a piece of steel plate is not immediately the most interesting or intellectually challenging activity. But okay, while the mechanics may nevertheless still generate a – although somewhat less positive – emotion, this is generally not true when it comes to packaging. Who in heaven's name lies awake over that? In the best case there are YouTubers such as Marco Reps [2] or Dave Jones from EEVBlog [3], who, by their own accounts, earn a living by opening packaging on YouTube using a ridiculously large knife. I have to humbly admit that the video by Marco Reps in particular, has touched my feelings deeply. You should know with how much care and love I packed up those kit sets.

Packing up products and kits sets looks deceptively simple, but nothing is further from the truth; it requires a significant amount of thinking and puzzling effort. Does the packaging need to have an attractive appearance or is that less important? Which box and which material and what exact size? How much is it allowed to cost? Obviously, the box – particularly in the case of fragile items – may not be too small, but also not too large, because you may need more filler material or larger inserts. The contents may not shift around inside the box when the box handled, because that could damage it and it is also not the intention that the customer can guess the contents by shaking the box. The latter is not only fashionable with the somewhat younger demographic but is also rather *in vogue* during the Christmas period [4]. For electronics there are further requirements such as antistatic packaging materials as well. When a product consists of multiple parts,



it is worth the effort to package it in such a way that the customer, when unpacking the box, has the parts immediately in the correct order for mounting, inverse tetrismatics at its best therefore.

Sometimes less obvious factors also feature in the choice for a particular type of packaging material; when I gave instructions to package the sand for the sand clock in an antistatic bag this initially resulted in quite a few raised eyebrows. No one had realised that dry fine sand will stick stubbornly to ordinary plastic foil because of electrostatic attraction, which makes sealing the bag more difficult.


Weighty Affairs

The packing of countless small parts, as is the case with electronics kits sets, comes with its own set of challenges. It is worthwhile to make a packing list and to sort the parts by type and function in individual small bags, which then go into a larger bag. This gives you a better overview and the opportunity for mistakes is considerably reduced, compared to just throwing everything in a pile into the box. As a final check, it is not a bad idea to weigh the bags individually on a precision scale. If a component is missing, or there is one too many, you will notice that immediately.

The assortment of scales and bags that we have for this purpose would be the envy of many a dealer in mind-expanding herbs and powders, but in addition, the packing table, empty containers, tape dispensers, sealing equipment, tubular foil, a label printer and industrial racking are darned handy too.

Another annoyance again is our old faithful postal service. For one reason or another, the cost of sending a package suddenly nearly doubles above a certain weight. The art is therefore to stay below this magical limit and I have struggled multiple times and for many hours with a scale and packaging materials to finally step into the post office, still without a great deal of confidence. Inverse tetrismatics on steroids therefore, something that can become quite tiring after a while.

Finally, there is fortunately still the Far East with Uncle Ali and consorts. Thanks to their

democratically low prices, relentless competition and other dubious economic activities, we for some time now, have had no more need to pack up kit sets. This saves me a whole heap of work, I would say. Thanks a lot for that. 

210625-01

Contributors

Text: **Ilse Joostens**

Editor: **Eric Bogers**

Translation: **Arthur de Beun**

Layout: **Harmen Heida**

Questions of Comments?

Do you have any technical questions or comments prompted by this article? Send an email to the editor of Elektor via editor@elektor.com.

WEB LINKS

[1] Ilse Joostens, "Sand Clock," Elektor 1-2/2017: <https://www.elektormagazine.com/magazine/elektor-201701/40130>

[2] Marco Reps: Elektor 6 Digit Nixie Clock: https://youtu.be/ge_9CNiZZ_A?t=25

[3] Dave Jones EEVBlog: Elektor IV-22 VFD clock: <https://youtu.be/SyXiWNZs7l4?t=1733>

[4] DIY Hacks and How Tos: A Prank for People That Shake Their Christmas Presents: www.youtube.com/watch?v=ZeCjiEiPqAM



By Clemens Valens (Elektor)

The global microcontroller market is more diverse than many people think. Let's take a look at some of the microcontrollers and manufacturers that are not often seen in *Elektor*. You might find one or more of them useful in a future project.

The choice of microcontroller for an Elektor design is mostly based on the availability of low-cost software development tools and device programmers and the possibility for individuals to buy it. As a result, many electronics enthusiasts have a rather limited view of the microcontroller market worldwide. There is much more out there than the PIC, AVR, ARM or ESP devices we encounter so often in DIY projects. Let's have a look at some of the MCUs that live in your blind spot.

It All Started With Four Bits

Introduced in 1971, the Intel 4004 is said to be the first commercially produced microprocessor (more about it can be found in our Elektor Industry Special about 60 years of electronics [1]). It was a 4-bit device. Together with its support chips it formed the MCS-4 family. It was followed up by the MCS-40 family with 4040 CPU. Texas Instruments's first successful microcontroller (not a microprocessor), the TMS1000 from 1974, also was a 4-bit device that, like the 4040, found its way into many pocket calculators.

In a world where microcontroller manufacturers seem to strive for data words as wide as possible, 64 bit is not uncommon, you would be surprised by the number of 4-bit microcontrollers that are still being used today. But why? The answer is probably a mix of legacy, power consumption and cost reasons.

A 4-bit MCU can be built with fewer transistors than devices with a wider word width. Therefore, all other things equal, they consume less power which helps increasing battery life. Fewer transistors also means less space and so a 4-bit core can be crammed on a corner of a chip when space for a larger core is lacking. As a die it can be smaller too, saving costs (even though one might wonder how much).

High-volume applications like calculators, timers, clocks and watches, bicycle computers, toys and remote controls make use of 4-bit MCUs and have done so for many years. As manufacturers usually avoid modifying products that have proven

themselves in the field – if it ain't broke, don't fix it – this explains why there still is a market for such devices.

In case you would like to try out a 4-bit microcontroller, have a look at the NY... families from Taiwan-based Nyquest. Development tools can be downloaded for free (**Figure 1**). Other manufacturer examples are EM Microelectronic from Switzerland, CR Micro from China and Tenx Technology from Taiwan.

8051

Before ARM became the main MCU core provider for almost every semiconductor manufacturer on the planet, there was the 8-bit 8051. Created by Intel in 1980 as MCS-51, the core (**Figure 2**) was licensed to several competitors, and it found its way into a plethora of products. Many of these products or their derivatives, variants, and siblings are still being produced today, and, 40 years after its introduction, 8051-derivatives are actively designed into new products.

Furthermore, 8051 users have developed a lot of software and know-how during this period, and you don't throw that away just because a better microcontroller comes along.

Finally, the 8051 core has become very cheap, if not free, making it an interesting option for semiconductor manufacturers trying to create ultra-low-cost devices. They don't always mention it in the data sheet, but if it says something like "1T instruction cycle" you can safely assume an 8051 derivative. The original 8051 consumed 12 clock cycles (called '12T') for most instructions whereas the most modern ones only require one (hence the '1T'). Besides needing fewer clock cycles per instruction, program execution is also (much) faster as some of these modern devices run at frequencies of up to 450 MHz instead of the 12 MHz of the original. Therefore, modern 8051 MCUs make for cheap yet powerful 8-bit microcontrollers.

Today, the main inconvenience of the 8051 is probably its incompatibility with modern programming languages like C and C++ due to its weird memory structure. To get the most out of it you need a commercial toolchain from Keil or IAR or similar compiler pros. The popular GCC toolchain that has been ported to all sorts of microcontrollers has no support for the 8051. The free toolchain SDCC does a somewhat reasonable job but is far from perfect. Programming the 8051 in assembler is, of course, also an option. You prefer Pascal? Have a look at Turbo51 [2].

You may find 8051-based MCUs in low-cost high-quantity products like toys, PC keyboards and mice, toothbrushes, home appliances, remote controls, etc., mainly from Asia where the 8051 appears to be particularly popular. Silan, SiGma Micro, SinoWealth, Silicon Laboratories, Sonix, STC, and SyncMOS (listed only those starting with 'S') all make them.

If you too would like to have a play with a modern 8051 derivative, check out the CH55x family from WCH. They are cheap

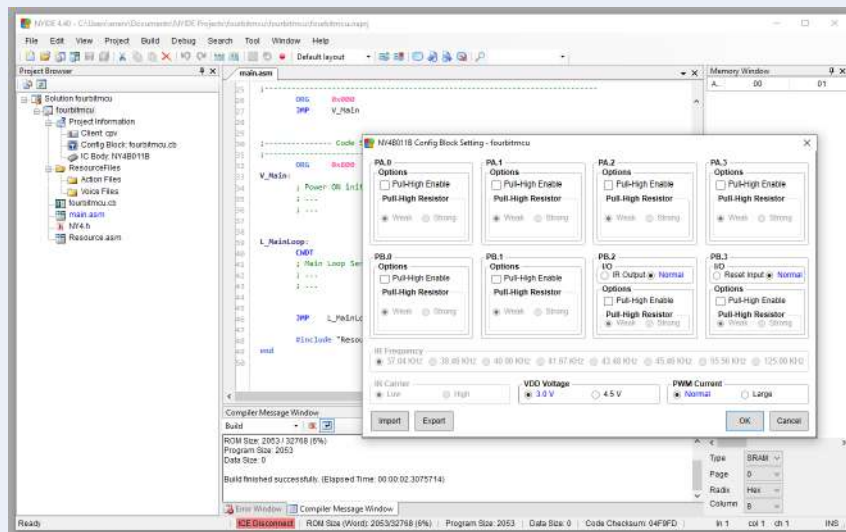


Figure 1: The NYIDE 4.40 from Nyquest shows that 4-bit microcontrollers can have modern IDEs too.

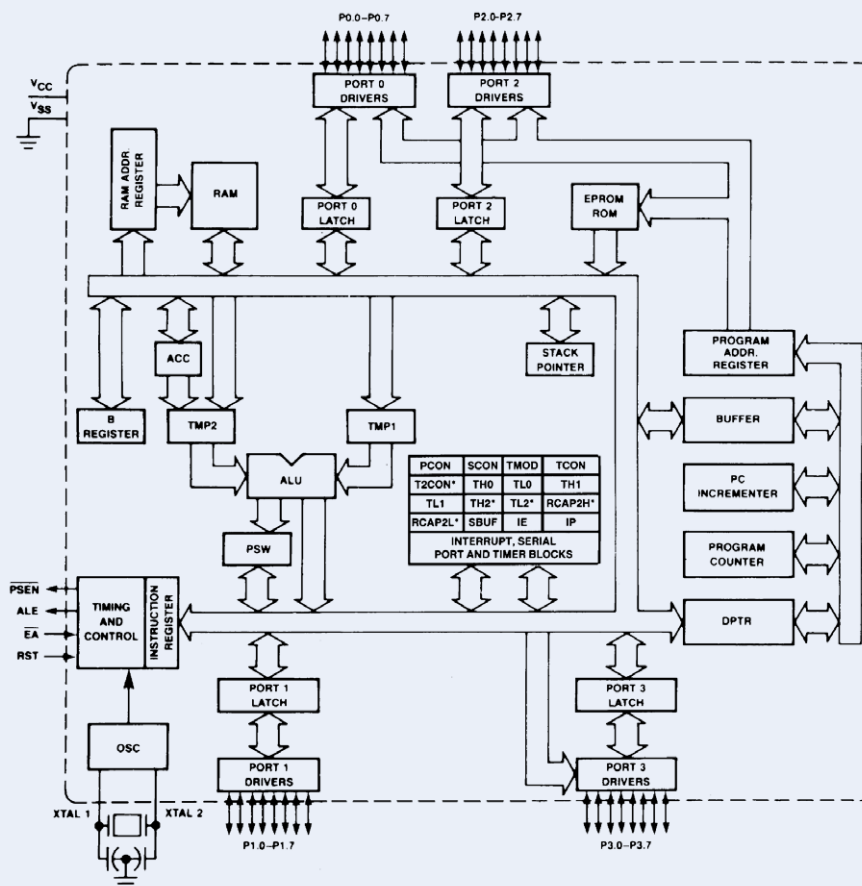


Figure 2: This is the basic architecture of what may very well be the world's most used microcontroller core, the 8051. (Source: Intel)



Figure 3: The Commodore 64, one of the famous home computers from the 1980s, has a 6502 inside. (Source: ralfsfotoseite @ Pixabay)

and documented in Chinese only, but they have a USB interface that allows easy device programming, and they are supported by open-source projects. Turn it into an Arduino? See for instance [3].

Golden Oldies & Die Hards

Like the 8051 from 1980, there are still some other processor cores around from that period, notably the 6502 and the Z80, but also the somewhat more recent 68000 by Motorola (now NXP). The Z80 and the CMOS version of the 6502, the W65C02, are still produced and actively supported by their creators Zilog and WDC respectively (WDC created the W65C02, not the 6502). The 68000, on the other hand, seems to be mostly kept alive to support legacy applications (but who knows how many companies have taken out a license?).

6502

The 6502 was used in several famous early computers like the Commodore 64, Apple II, and BBC Micro, and has been very successful indeed (Figure 3). Its improved and low-power CMOS version is still going strong even though they are rather expensive. The reason for this is that most users only license the core to use inside FPGAs, ASICs and similar custom chips. Since this is all confidential, it is hard to find out which devices are concerned.

However, WDC does produce packaged chips that you can try out. An example is the W65C265S microcontroller with a 16-bit W65C816S CPU that is fully compatible with the 8-bit W65C02S and that runs from as little as 1.8 V. Controller modules

exist too, and even a companion's board with Seeed Studio Grove, Sparkfun QWIIC, and MikroE Click connectors.

Z80

The Z80 is another highly successful processor from the late seventies, early eighties of the previous century. Zilog developed the core in 1975 and the company still produces it. It has been licensed to and was copied and cloned by many other manufacturers worldwide which has resulted in a huge user base. Several families have seen the light, like the eZ80 that runs from clocks up to 50 MHz, or the Z8 and eZ8 Encore!. The latter is found in, for example, the ZMOTION product line, a family of MCUs optimized for PIR motion detection.

If you want to get your hands dirty, you might want to have a go at the Z8FS040BSB with its 4 KB of flash memory and five GPIO pins in a convenient 8-pin SOIC package. Note that the free compiler SDCC supports several Z80-based MCUs like those from Rabbit (now Digi) and the Nintendo Gameboy.

Ultra-Low-Cost Devices

Many electronic products containing microcontrollers are produced in massive quantities. Think for instance about home appliances, clocks, electric toothbrushes, e-cigarettes, Corona virus testers, smart cards, smoke detectors, toys, etc. (Figure 4). To get an idea of the figures: gaming consoles like the Nintendo Gameboy, Wii and Switch have all passed 100 million units sold. Imagine what the numbers are for, say, smart cards. Saving a cent on the costs of such a product is huge and so there is a large market for ultra-cheap microcontrollers.

Some of these microcontroller manufacturers that have come to the attention of electronics amateurs are Padatak, MDT, and Holtek.

Padatak

Padatak makes 3-cent one-time programmable (OTP) and flash-based MCUs. The



Figure 4: A few examples of applications made possible by ultra-low-cost microcontrollers. (Source: Holtek.com)

particularity of their devices is the architecture based on FPPs (Field-Programmable Processing units). These are register banks with a program counter, stack pointer, accumulator and flag register that allow for fast context switching. This is useful for e.g. interrupt handling and multitasking. They remind a bit of the 8051's four register banks. However, as most of their products only have one FPP (some have two, the PFC460 has four, while the MCS11 has eight) they are just basic MCUs.

The PMS150 is a good place to start. An excellent write-up of these devices can be found at [4]. SDCC (once more) supports the PDK14 and PDK15 while support for the PDK13 is being worked on.

MDT

As said before, MDT a.k.a. Micon Design Technology makes (made?) clones or derivatives of Microchip's PIC devices. As PIC MCUs are popular amongst makers, MDT devices have attracted some attention. However, while doing research for this article their website suddenly went offline. Searching the internet for MDT devices produces several results for MCU cracking and reverse engineering services of, amongst others, MDT devices, suggesting that they are widely used.

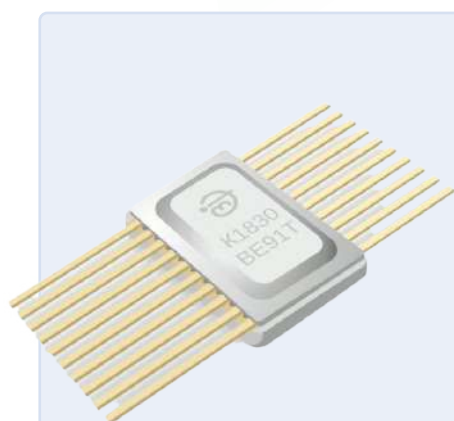


Figure 6: The K1830BE91T from NIIET has an 8051 core and is functionally equivalent to Microchip's AT89C2051. (Source: <https://niiet.ru>)

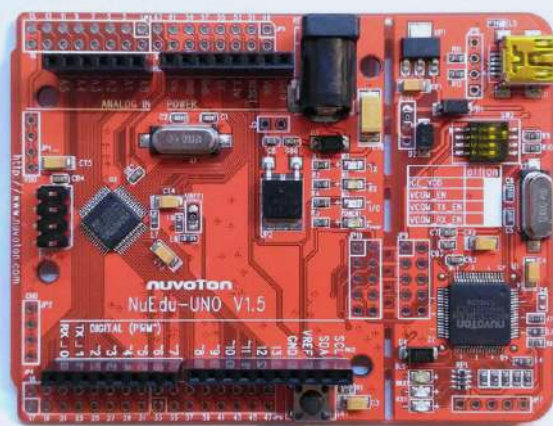


Figure 5: The Nuvoton NuMaker Uno board has a detachable NuLink programmer/debug module. (Source: <https://danchouzhou.blogspot.com>)

Holtek

A few Holtek products have been used in the past in Elektor projects. These were not microcontrollers but keyboard- and RC channel decoders. However, they also make MCUs, and lots of them too, from 8051 types to ARM Cortex-M0 and M3 and devices based on their own core. As many of the low-cost MCU providers do, the Holtek product line too is divided in application-specific and general-purpose ("I/O-type") MCUs. Documentation is good and the HT-IDE3000 IDE (assembler and C) is free (although somewhat hard to find), but a Holtek programmer is needed.

A device I found interesting is the HT66F4550 analog MCU which integrates two opamps and has an audio output.

What About the Big Ones from Asia?

Elektor has published hundreds of microcontroller-based projects, and most of the time they contained a PIC or AVR device from Microchip (and formerly Atmel), an ESP device from Espressif or an MCU with an ARM core from NXP or ST. The MSP430 from Texas Instruments also made a few appearances. Except for Espressif, all these manufacturers are European and American. This shows how biased we are as there are many big Asian companies that produce MCUs.

Renesas

One of the largest, if not the largest Asian semiconductor manufacturer is Renesas, built out of NEC, Hitachi, and Mitsubishi divisions. Some even claim that it is

the world's number one MCU provider. Long-time Elektor readers might remember the R8C and R32C/111 article series from some 15 years ago [5]. The recent RX671 family of 32-bit microcontrollers specializes in fast real-time control and contactless human-machine interfacing (HMI) by proximity switches and voice recognition, a perfect fit for modern hygienic HMI designs. Renesas also provides a large number of other development and evaluation boards, and I encourage you to give them a try and report your findings.

Nuvoton

Spun off from Winbond in 2008, Nuvoton acquired Panasonic's agonizing chip division in 2020. They have a large offering of 8051- and ARM-core MCUs, and also a few proprietary core devices. Contrary to many competitors, Nuvoton does not have their own toolchain. For the 8051 devices, they rely on Keil and IAR while the ARM parts use Eclipse. On GitHub you can find support for using SDCC with some Nuvoton devices.

The NuMaker Uno (Figure 5) is a good way to get started with Nuvoton. It is an Arduino-compatible board with a NuMicro NUC131 ARM Cortex-M0 controller. It also comprises a detachable Nu-Link debugger/programmer module that can be used with other devices as well. Software support can be found on GitHub (OpenNuvoton). Check out the NuMaker repository for support for mbed, Arduino, MicroPython and more.

Russian MCUs?


To complete this article, I wanted to add some information about Russian micro-



controllers. Unfortunately, I do not read Russian, yet most websites are in Russian, and this makes finding useful information difficult. I did come across Milandr, Mikron and fabless Syntacore who are all doing RISC-V-based controllers. According to [6], Milandr also has a license for ARM cores, but their website doesn't mention such parts.

NIJET produces the K1921VK01T which is targeted at motor control and smart metering applications and is built around an ARM Cortex-M4F core. OpenOCD has support for this MCU. In October 2021 NIJET announced a RISC-V-based controller to replace the STM32- and MSP430 parts that are currently being used in "civilian equipment" (as they call it) in Russia. They also have 8- and 16-bit RISC MCUs and a few MCS-51 (Intel 8051) and MCS-96 (Intel 80196) devices (**Figure 6**).

A World of Options

This article presented a few microcontrollers and manufacturers that are not often seen in Elektor projects, yet they represent an important part of the global MCU market. Of course, this article is far from complete, and some interesting devices or manufacturers may have been missed. While researching this article, I compiled a list of more than 50 active microcontroller manufacturers, and I am sure there are many more. If you know of other unknown yet interesting devices that you would like to share with the other readers, please let me know. 

210630-01

Contributors

Idea & Text: **Clemens Valens**

Editor: **Jens Nickel and C. J. Abate**

Layout: **Harmen Heida**

Questions or Comments?

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



Both the 4004 and 4040 were designed by Federico Faggin. Which other famous processors did he design as well?

Intel 8080, Z80, 8085, 8088



RELATED PRODUCTS

- > Book: *Mastering Microcontrollers Helped by Arduino* (SKU 17967)
www.elektor.com/17967
- > Raspberry Pi RP2040 Microcontroller (SKU 19742)
www.elektor.com/19742
- > Book: *ARM Microcontroller Projects* (SKU 17620)
www.elektor.com/17620

WEB LINKS

- [1] Stuart Cording, "The Birth of the Microprocessor," Elektor Industry 03/2021: www.elektormagazine.com/magazine/elektor-241/60042
- [2] Pascal for 8051: <https://turbo51.com/>
- [3] CH55xduino: <https://github.com/DeqingSun/ch55xduino>
- [4] Padauk PMS150: <https://jaycarlson.net/2019/09/06/whats-up-with-these-3-cent-microcontrollers/>
- [5] Gunther Ewald, "The R8C Family," Elektor 01/2006: <https://www.elektormagazine.com/magazine/elektor-200601/18155>
- [6] Russian microcontrollers: <https://geek-info.imtqy.com/articles/M4836/index.html>

Monitor and Debug Over the Air

A Solution for Arduino, ESP32 and Co.

By **Peter Tschulik** (Austria)

Modern microcontroller boards, equipped with an ESP32 or an ESP8266, for example, offer a lot of performance for not much money and a high level of programming convenience especially via the Arduino IDE. The over-the-air (OTA) feature is particularly interesting. It enables both program updates and data to be uploaded conveniently via a bootloader using Wi-Fi communication, without the need to physically connect the board to the USB port of a PC or laptop. However, to transmit serial data (such as debugging information) wirelessly, we need another solution.



The Wireless UART-RS232 long distance set.

IoT devices operating remotely 'in the field' benefit from the ability to perform software updates over the air (OTA). A good tutorial on the subject using the Arduino IDE can be found at [1]. However, this method has one drawback: the serial monitor is not supported, so that output and debugging information cannot be transferred.

On my last project, a garden watering system based on the ESP32, there was a problem with a malfunctioning sensor. Instead of

laying a USB cable all the way across the terrace into the living room and to my computer, I started looking for a wireless solution. Despite an intensive search, I couldn't find an off-the-shelf module that would do the job according to my requirements so I set about developing my own solution. I seemed to remember systems that enabled wireless transmission of serial interface data; in fact I quickly found what I was looking for from a company in the Far East. The Wireless UART-RS232 long

distance set [2] offers transfer rates of up to 115,200 baud with configuration options and a range of up to 1 km. With a price of £50 for the sending and receiving pair, I thought this represented a good price/performance ratio for this project.

A USB-Host Adapter

The selection of a USB/serial converter turned out to be much more difficult. Although it would be possible to connect the serial data signal from the remote watering controller directly to

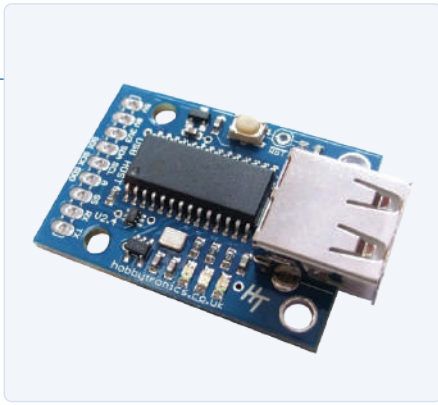


Figure 1: Good things come in small packages. The USB host board.
(Source: www.hobbytronics.co.uk)

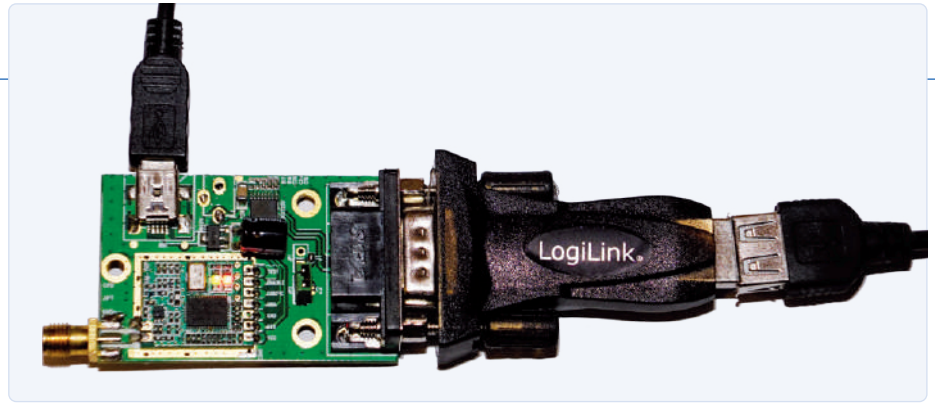


Figure 2: The Wireless UART-RS232 long distance modules need to be configured. In the picture you can see the jumper in place and the green and red LEDs.

the serial transmission system, I wanted a more universal solution using a USB port connection so that other types of development board could also easily use the link. For this you need to provide a so-called 'USB host adapter' which plugs into the USB port at the remote site and talks to the development board as a PC would. I struck lucky when I spotted the 'USB-Host Shield' for Arduino after some intense internet trawling [3] and [4]. The item was stocked by a local supplier so I put in an order and within a few days I was busy unwrapping it at my desk. It did not take too long for me to discover a problem. There are many different development platforms out there in the maker world and several different brands of USB interface chips. The ESP32 boards for example use the CP210x or FTDI chips, Arduino boards use the Atmel controller or FTDI chips. Unfortunately, the libraries for the Arduino USB Host Shield can only support one type of chip at a time. That means, you would need to make the host shield software configurable and find out which flavour of chip is fitted to the development platform you are using. Hardly a plug-and-play solution! Eventually, I found a USB-host interface board from the UK which was much more accommodating of different brands of interface chip (**Figure 1**) [5].

What is particularly interesting about this board is that it can work with a wide range of devices such as flash memory sticks, USB keyboards, USB joysticks, USB mice, PS3/PS4 dual shock controllers, and has serial drivers for FTDI, CP210X, PL2303, CH340/1 and CDC, MIDI devices and USB modems. A list of all the free firmware applications is given on their web site. If you know exactly the application you want to use the board for, you can specify it when ordering and it will be supplied with the firmware installed. For this application, I

needed the *USB Host - Serial Driver for FTDI, CP210X, PL2303, CH340/1 and CDC* firmware [6]; it supports all the relevant USB chips used by the different development boards.

For programming and configuring the wireless UART RS232 modules or the USB host board, at least one USB to serial converter is required, which can either be obtained from the same manufacturer of the wireless UART RS232 modules [8] or, for example, from Logilink [9]. For my project, I needed two RS232 level converters. I chose the good old MAX232 [7] solution.

Setting Up the Link

When I received the Wireless UART-RS232 Long Distance set, it was not clear to me how I should actually configure the modules. I contacted the seller and the next day received an email with the software, a manual and a link to a configuration video [10]. First, I had to take the modules out of their plastic cases and solder a pin header to connector position J2 (a row of pads just in front of the RS232 connector). I then put jumpers on the pins according to the instructions so that the module enters the configuration mode. Next, I connected the module to the PC via the USB/serial adapter. The module is powered from a 5 V power supply (e.g., a USB plug-in power supply) via the mini USB connector (**Figure 2**).

If both LEDs light up, you can fire up the *HY-TRP Setting GUI.exe* program and set the correct COM port and the default baud rate to 9,600. Then click on the *Open COM* button and read out all values with *Read All Settings*. The values are set as shown in **Figure 3**.

Further details can be found in the manual [10]. Some issues of link reliability were resolved by reducing the transmission link baud rate down from 115,200 to 57,600. The USB host board has a buffer that tolerates different transfer speeds. Both of the modules must be configured identically. Lastly, when configuration is complete don't forget to remove the jumpers.

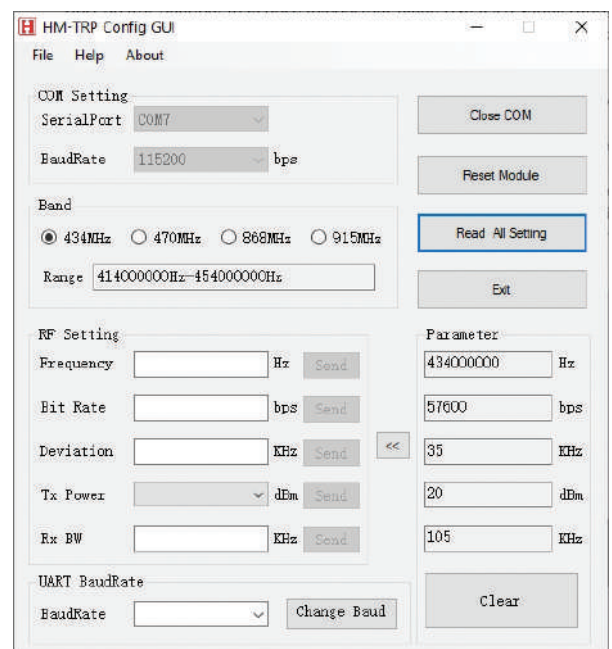


Figure 3: The recommended configuration for the Wireless UART-RS232 long distance module.

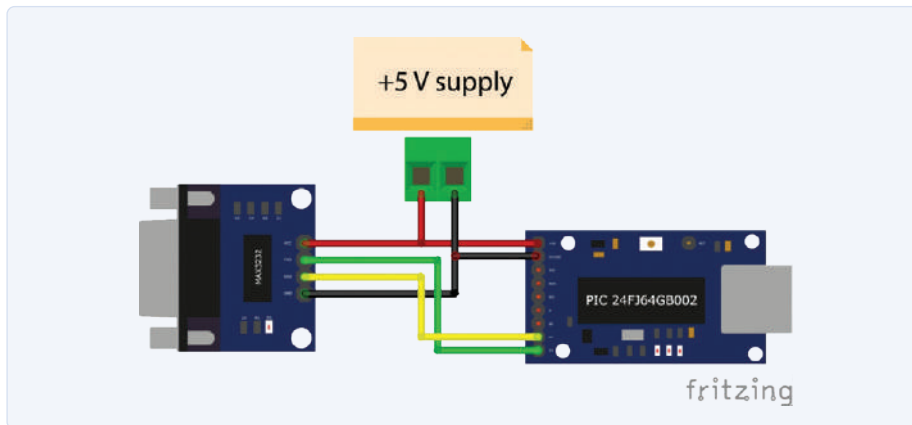


Figure 4: Hook up of the USB host board for configuration.

USB Host-Board Configuration

To program the USB host board with the correct firmware, please refer to the description given at [5]. There are many programs available for download from Hobbytronics which can be

flashed to the board. The correct firmware can be loaded onto the board using the instructions given at [6], and there you will also find a description of the parameters of the command interface. For configuration, the USB host board

has to be connected to the MAX3232 level converter board as shown in **Figure 4**. The RS232-TTL converter connects to the RS232-to-USB adapter, which in turn connects to the PC via a USB cable. The circuit now only needs power from a stable 5 V supply.

Now you can start a terminal program (here, for example, Hterm), select the correct USB port and set the baud rate to the default value of 9,600 baud. Clicking on the **Connect** button at the top left will connect you to the USB host board. At the top right **Newline at** and below the main window **Send on Enter** options are changed to **CR + LF**. If you now enter **HELP** in the input field of Input Control field below the main window, the current parameters should be listed as shown in **Figure 5**.

The parameters for our application are now modified using the input field of **Input Control**. First, we can enter **BAUD 57600** command to change the baud rate which must match the

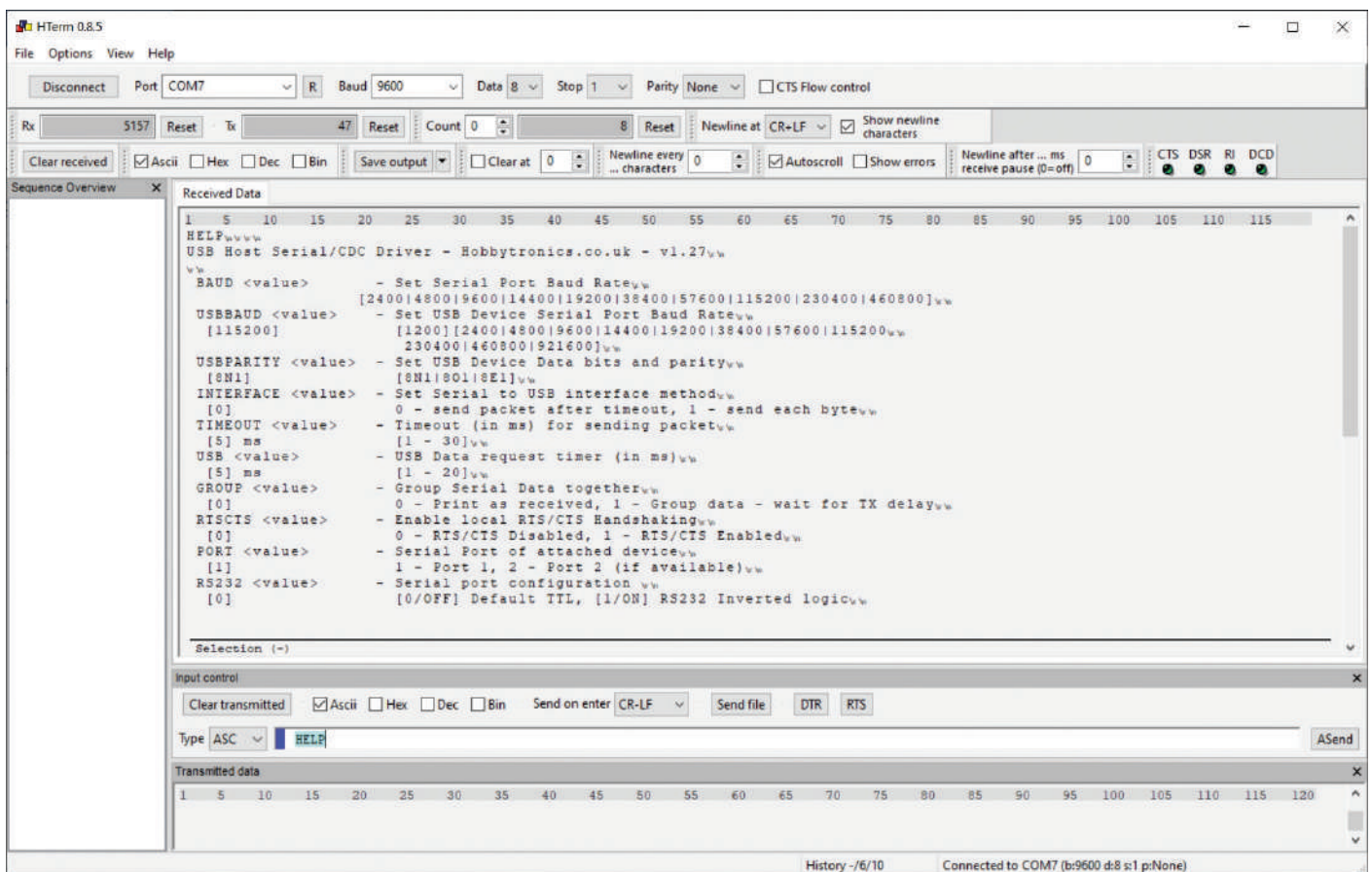


Figure 5: Configuring the USB host board using Hterm.

Table 1: Parameter setting in HiTerm

BAUD	57600
USBBAUD	115200
USBPARITY	8N1
INTERFACE	0
TIMEOUT	5ms
USB	5ms
GROUP	0
RTSCTS	0
PORT	1
RS232	0

baud rate set for the *Wireless UART Systems*. If you use *Disconnect* then *Connect*, to disconnect and reconnect the terminal to the system you should be able to check the communication link with the USB host board using *HELP*.

The *USBBAUD 115200* command will set the USB baud rate to 115,200. This means that the serial interface of the connected Arduino/ESP32/ESP8266 board in the Arduino IDE must always start with the command *Serial.begin (115200)*.

Default settings of *USBPARITY*, *INTERFACE*, *TIMEOUT*, *USB*, *RTSCTS*, *PORT* and *RS232* are used. The default value of the serial data grouping option is OFF. The command *GROUP 0* also sets these properties to their default value. **Table 1** gives a summary of the correct settings. This completes the configuration.

Wiring the Modules

Before the transmitter (at the IoT board in the field) and receiver (at the PC end) can be wired up it will be necessary to remove the PCB mounted power supply barrel socket (next to the USB socket) on the Wireless UART-RS232 long distance Modules so that the pads are accessible.

On the transmitter side, the mini-USB socket of the USB host board will power the unit. **Figure 6** shows the wiring at the transmitter end. To connect the *Wireless UART-RS232 long distance module* to the RS232/TTL

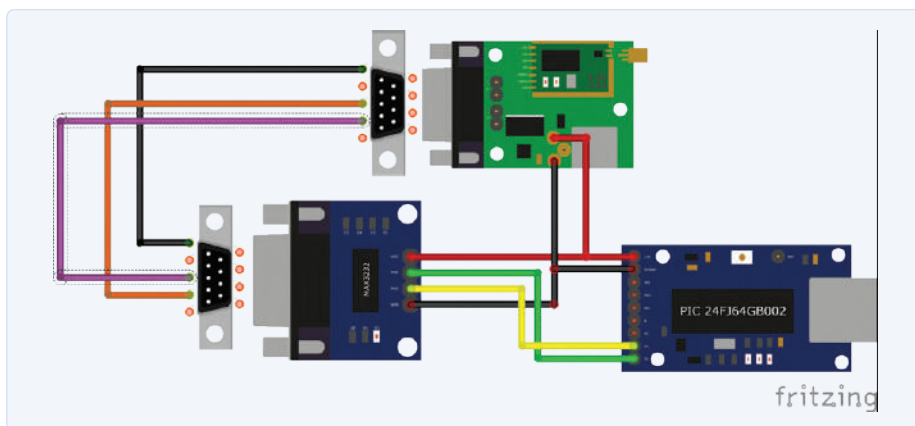


Figure 6: Remote transmitter wiring (IoT board location). The green board is the wireless module. The IoT board supplying data connects to the USB port bottom right.

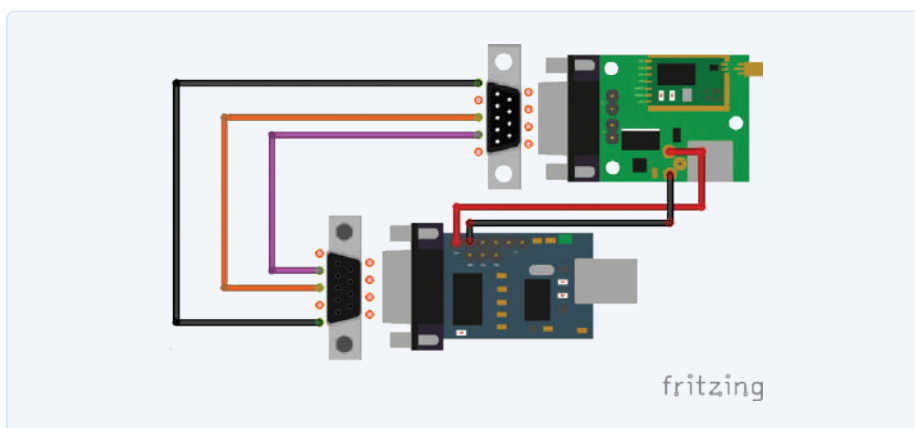
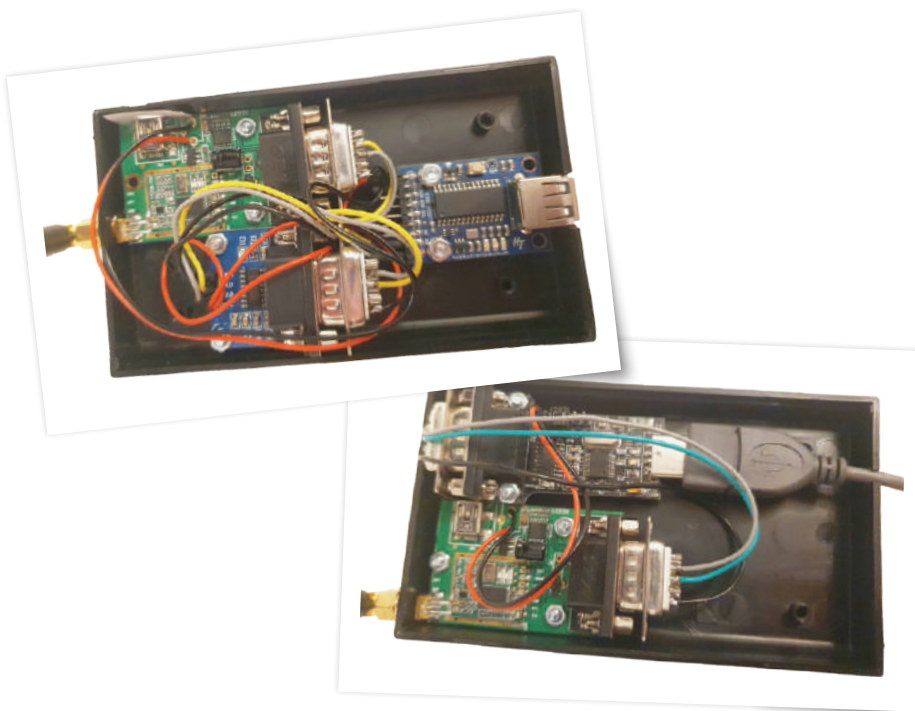


Figure 7: Receiver station wiring. PC connects bottom right via a USB extension cable.

adapter, you need to use a short cable terminated with male SUBD-9 connectors. Wires to Pin 2 and Pin 3 are switched over in the cable as shown in the picture.

The compact module from [8] is used as the RS232/USB converter for the receiver and is wired up according to **Figure 7**. I used a USB extension cable between the PC and receiver unit to give some flexibility in positioning the unit indoors.

Finally, I installed the transmitter and receiver unit into two matching enclosures. The tests of my OTA system showed consistent transmission through two floors with reinforced concrete ceilings. Altogether the system provides a convenient and reliable long range 'virtual USB cable' facility useful when developing and debugging remote IoT applications. ◀

200549-01

Questions or Comments?

Do you have any questions or comments about this article? Email the author at peter.tschulik@chello.at or contact Elektor at editor@elektor.com.

Contributors

Idea, Design and Text: **Peter Tschulik**
Editor: **Rolf Gerstendorf**
Translation: **Martin Cooke**
Layout: **Giel Dols**

Operating Instructions for Practical Use

- › Configure the serial communications interface of the board used to 115,200 baud using `Serial.begin(115200);` in the Arduino IDE.
- › Once the firmware has been transferred to the remote board, supply the transmitter unit with 5 V via the mini USB socket of the wireless UART-RS232 long distance module. Then connect it to the remote board using a short USB cable.
- › Plug the receiver unit into a free USB port on the PC. The receiver is powered from the PC.
- › Start the Arduino IDE, select any board, select the receiver board USB interface and start the Serial Monitor.
- › Any data sent by the board should now be displayed.



RELATED PRODUCTS

- › **Bundle: The Complete ESP32 Projects Guide + ESP32-DevKitC-32D (SKU 19897)**
www.elektor.com/19897



- › **ESP32 & ESP8266 Compilation (PDF) (SKU 18516)**
www.elektor.com/18516

WEB LINKS

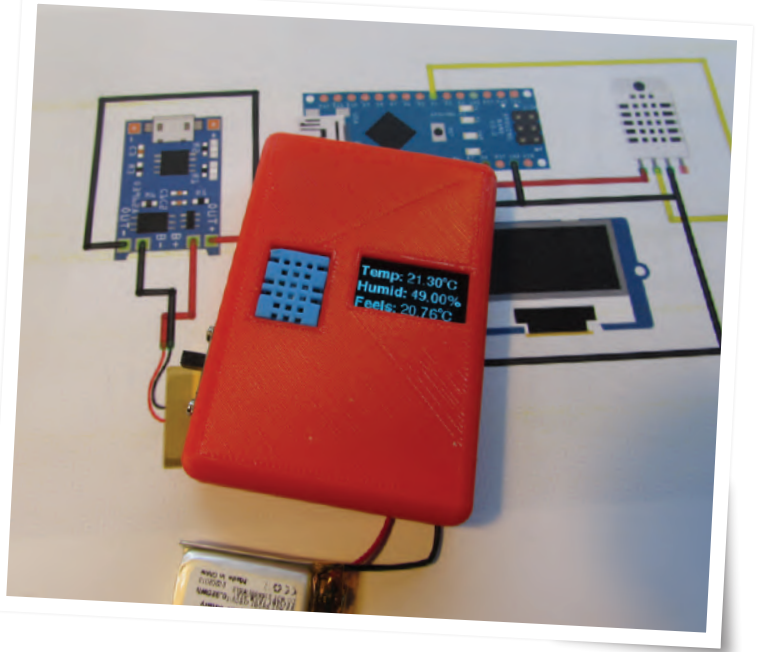
- [1] ESP32 Basic Over The Air (OTA) Programming In Arduino IDE: <https://lastminuteengineers.com/esp32-ota-updates-arduino-ide/>
- [2] Wireless UART-RS232 long distance Module: <http://www.aliexpress.com/i/32242647632.html>
- [3] SparkFun USB Host Shield: <http://www.sparkfun.com/products/9947>
- [4] USB-Host-Shield für Arduino: <https://bit.ly/3fJQrTO>
- [5] USB Host Controller Board V2.4: <https://www.hobbytronics.co.uk/usb-host-board-v24>
- [6] USB Host - Serial Driver for FTDI, CP210X, PL2303, CH340/1 and CDC: <https://www.hobbytronics.co.uk/usb-host-serial>
- [7] RS232-TTL-Adapter: <https://bit.ly/3peJH3r>
- [8] 2 Functions USB COM Port DB9 TTL232 RS232 TTL 232 CH340T USB2.0 Cable Adapter: <https://bit.ly/3fHUGQ3>
- [9] LogiLink USB-2.0 to Serial-Adapter: <https://bit.ly/3ma21Kg>
- [10] Configuration instructions for the Wireless UART-RS232 long distance module HY035_HM-TRP-RS232.zip: <https://1drv.ms/u/s!AjrAGEbCBLsugxiqmd1FRRCJjic>

Portable Temperature- and Humidity-Measuring Device

Using Ready-Made Modules

By Aarav Garg (India)

One of the easier approaches to modern electronics is to build projects by combining ready-made modules. Only some wiring and — in most cases — software is needed to create a new device, like the one we present in this article.



Luc Lemmens, Elektor Lab: The designer is the 15-year-old innovator Aarav Garg from India, who is more than happy to present his so-called "Pocket Weather Station" in our magazine. It is constructed using an Arduino Nano, a 0.96" OLED display, a DHT11 humidity and temperature sensor. It is powered by a LiPo battery that can be charged via a USB charging module. A tailor-made, 3D printed enclosure completes this handy, portable device. We will let Aarav speak for himself to tell how he came to this design. I have built and successfully tested it in the Elektor Lab, some additional instructions and remarks can be found in the text boxes.

In this article, you will learn how to build a pocket weather station using an Arduino Nano board. It will be a compact device that you can carry anywhere, right in your pocket; and it will be capable of displaying the live temperature and humidity on the OLED display present on it. This is a great self-care device as you will always know when to take an umbrella out with you, both for the rain and the scorching heat! The device has a built-in rechargeable 160-mAh LiPo battery. It is a really great project for learning and is also fun to make.

Step 1: Gather Components

The first thing to do when beginning with any project is gathering the required components,

as shown in **Figure 1**. The required components for this project are:

- Arduino Nano with cable
- DHT11 Temperature Sensor Module
- 0.96" OLED Display
- TP4056 Battery Charging Module
- Small Battery (I used a 160-mAh LiPo battery)
- Slide Switch

Tools:

- Soldering Iron
- Jumper Wires
- Hot Glue Gun
- 3D Printer for enclosure (optional)

Collect and/or buy all the required stuff and move on to the next step.

Step 2: Arranging Components

Now we need to plan the position of all the components inside an enclosure. I wanted to keep the device as thin as possible, so it is actually convenient to carry in a pocket. Thus, I spread all the components out and did not go with a multiple component-layer structure as that would decrease the X and Y dimensions of the device but eventually increase the Z dimension, which is nothing but the thickness of the device.

Refer to **Figure 2** to check how I stacked the components inside my pocket weather

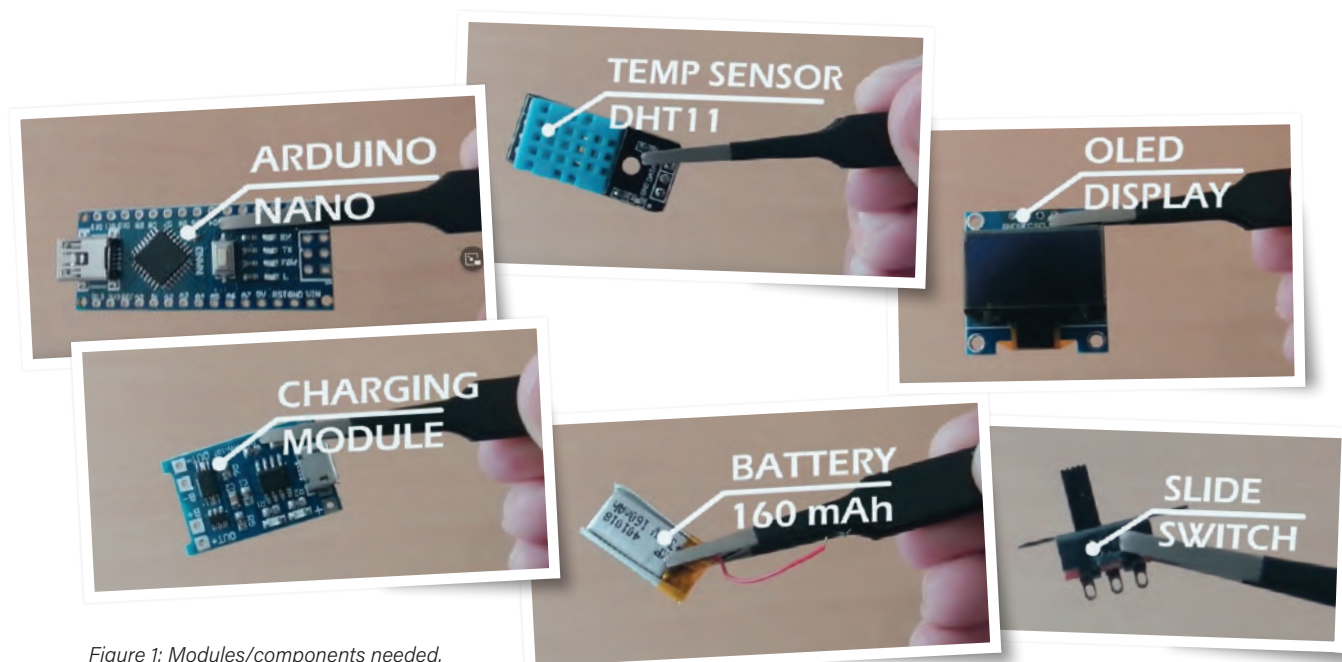


Figure 1: Modules/components needed.

station. Or, you can also come up with your own ideas.

Step 3: Schematic

Now we need to draft out a schematic for our pocket weather station. The schematic for this project is very simple as there are very few modules which have to be just connected together without the need for any exter-

nal modifiers. The schematic diagram can be found in **Figure 3**. You can refer to it if required. We need to connect the battery to the battery charging module and the output of the battery charging module to the Arduino Nano board. I have used an Arduino Nano board because of its size which is perfect for this project! Next, connect the temperature sensor module and the OLED display

to the Arduino board. After completing the schematic, move on to the next step.

Step 4: Soldering/Making Connections

Now, we just need to solder all the components together according to the schematic that we previously drafted. Try keeping the length of the wires cut-to-fit to prevent a mess

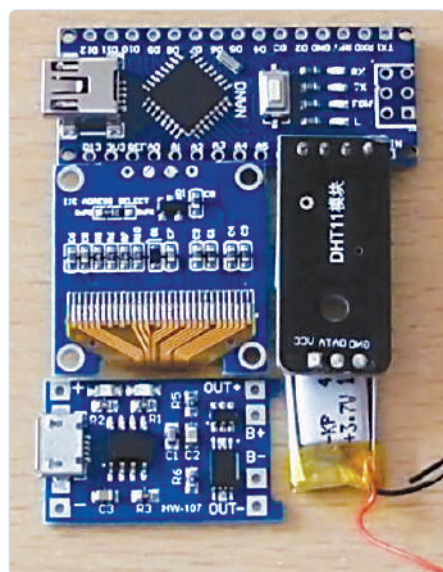


Figure 2: Arranging the modules.

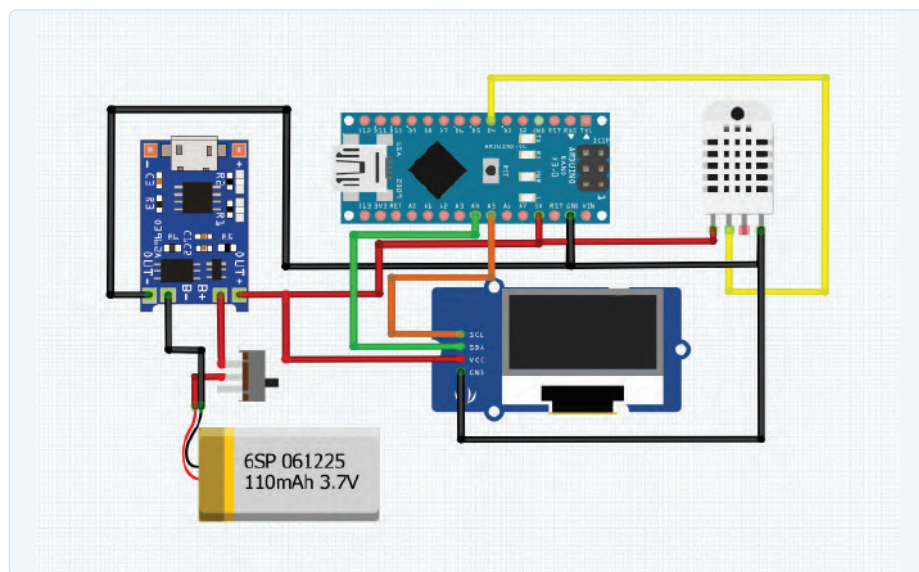


Figure 3: Fritzing diagram showing interconnections.

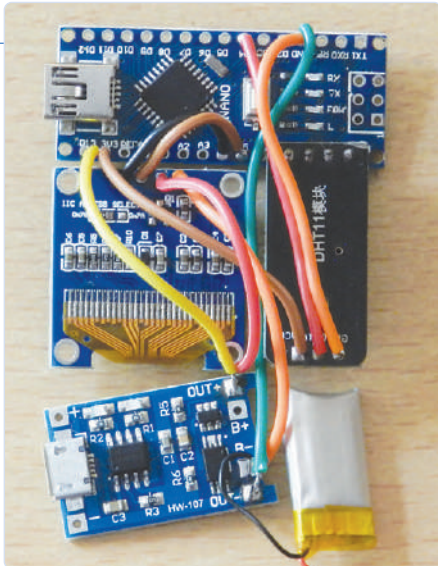


Figure 4: Most of the interconnections made.

of wires later on. Also, try to solder everything accurately to prevent any kind of short circuit. It might be a tedious process but believe me later on it will feel worth the effort.

After you are done with the soldering, it should look something like the construction shown in Figure 4. Just move on to the next step.

Step 5: After Soldering

I am sure we can't carry this wired mess around like this, and so it is pretty clear that we need an enclosure for our pocket weather station to give it that proper professional look.

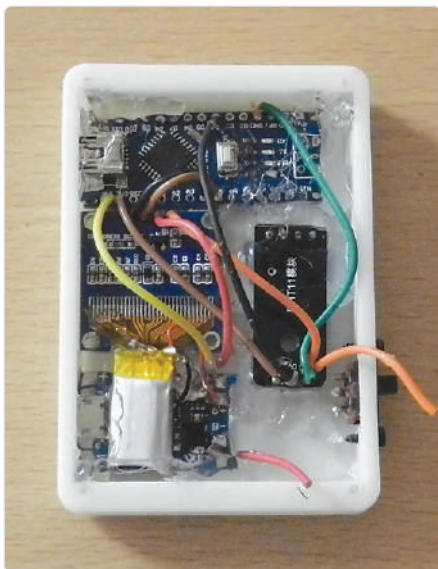


Figure 6: The electronics hot-glued inside the enclosure.

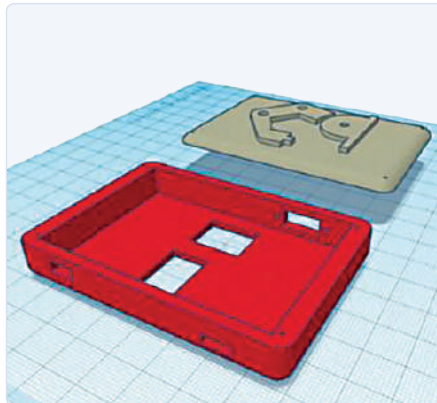


Figure 5: Design of the enclosure, made in Tinkercad.

And the best option that we have here is 3D printing. So, move on to the next step to design the enclosure and print it!

Step 6: Making Enclosure

I went ahead and designed the enclosure for the pocket weather station (Figure 5) in Tinkercad, which is an amazing CAD software. It supports all skill levels, so if you are a beginner at CAD, it shouldn't be a problem. I do not have a 3D printer, so I went to IAmRapid, uploaded my .STL files for an instant quote, and ordered the parts right away. The enclosure has a great build quality.



Figure 7: The switch is attached to the outside of the enclosure.

Fortunately, all the cut outs that I had made in the design for different modules and ports turned out to be 100% perfect. The 3D files for printing the enclosure are available for download at the [Elektor Labs](#) page for this project [1]. Let's move on to placing the entire circuit inside the enclosure and fixing it in place.

Step 7: Placing Circuit in Enclosure

Now, we need to place the whole circuit inside the enclosure that we earlier designed and 3D printed. It is very important that all the ports go in their respective cut outs to give the device the much needed professional look. Also, it is important that all the components are firmly fixed in their place and do not move inside the enclosure to ensure proper and smooth functioning of the device! I used some hot glue for fixing the circuit in the proper way, Figure 6 shows the result. After you are done with this, move on to the next step.

Step 8: Adding the Switch

Now after you are done placing the circuit inside the enclosure, it is time to add the slide switch in its dedicated slot. We did not connect the switch previously in the circuit itself because the switch needs to be inserted into the enclosure from the outside (Figure 7).

After inserting the switch into its slot, use two small screws to fix it in place. Then connect the two wires, one from the VCC of the Arduino board and one wire from the positive output of the battery charging module. In this way, we will be able to turn the complete circuit on when we turn the switch on.

Step 9: Closing the Enclosure

Now, we need to close the enclosure. I used some screws to fix the lid of the enclosure in place. I had already made screw holes in the design of the enclosure so there is no problem later on!

Just make sure that the lid is in place so the device even looks professional and is convenient to carry! I have put my logo on the lid design to give it a more aesthetic and customized look.

Step 10: Coding Battle!

Now, we need to do something very important and that is coding our pocket weather station because without the code running in

Sensor Data

The DHT11 is a well-known humidity and temperature sensor in the maker scene. It is not the most accurate device, but affordable and readily available. The temperature and humidity readings from the sensor are

displayed and used to calculate the so-called Heat Index [2], the third value displayed at the bottom of the LCD (labeled with 'Feels:'). It is the temperature perceived by a person at a particular combination of temperature and relative humidity; it is calculated in the

Arduino DHT11 library. This should not be confused with the — somewhat better known — *chill factor*, where the influence of wind speed is also included in the calculation.

The Arduino Sketch

With projects like this, nothing works without the software, in this case the sketch for the Arduino Nano. As for many popular modules, integrated circuits and sensors there are ready-made Arduino libraries available to make the life of the programmer a lot easier. In Aarav's sketch — among others — libraries for the graphics and control of the OLED display, and reading and processing of the DHT sensor data are used. The listing below contains the `setup()` and `loop()` functions of the sketch, showing that only a minimal amount of code is needed to make the weather station work, most of the processing is done within the functions that are provided in the libraries. Two statements are needed to read the temperature and humidity from the DHT11, and a third to calculate the heat index from the sensor data. The rest of `loop()` is used to show the resulting values on the display. The larger part of the sketch that is not shown, is needed to display the author's logo on the screen when the device is powered on or reset. The sensor data are also sent to the serial monitor of the Arduino IDE for debugging purposes.

```
void setup() {
  Serial.begin(9600);
  dht.begin();
  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 128x64
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // don't proceed, loop forever
  }
  testdrawbitmap(); // draw the required image
  delay(1000);
}

void loop() {
  float h = dht.readHumidity();           // read humidity
  float t = dht.readTemperature();        // read temperature
  float hic = dht.computeHeatIndex(t, h, false); // compute heat index for temp in °C
  // Printing the results on the serial monitor
  Serial.print("Temperature = ");
  .....
}
```

Troubleshooting, If Things Don't Work

If you are here, then I assume that you have built the project. It either worked which is great or it didn't which is also great because you not only will know how you build something but also how you do not build it and this kind of learning is very useful.

LED display blank: You have either burnt out your OLED display or maybe there is an error

in your code, for example you might have forgotten to initialise the display or incorrect I²C address for the display. Recheck your code and try to check whether the display itself is working or not. If it is, then try tweaking the code.

All readings are "NA": This will happen if your temperature sensor has an issue. It could be that you have done an error in connecting the temperature sensor to the

Arduino board. Just recheck the connections. If the connections are right, it is probably true that you have an issue with the sensor itself. Try replacing the sensor!

Works with USB cable but not battery: If this happens, then there is a problem with your battery or maybe the connections of the battery!

Hardware, Notes from Elektor Lab

For the compact construction of the Pocket Weather Station that Aarav built, you'll need to have an Arduino Nano without pin headers but unfortunately: the Nano boards from my drawer both had pins. This makes the modules too high to be built into a flat enclosure and it is less easy to solder connecting wires to them. Of course, you can shop around for a module without pins, but with a little care pin headers can be removed without damaging the board. Cut the pins off to the plastic base using a small side cutter, then cut away the plastic. Finally, solder the remains of the pins out of the board and remove the solder with a desoldering pump and/or desoldering braid. The 6-pin ISP-header will be a bit more of a

challenge, but it can be removed using the same method.

The 0.96" display I ordered also had a pin header, I removed that too to save space inside the enclosure.

But with these displays, there may be another issue: most of these can be configured for either an SPI or I²C interface, the latter is needed for this project. I made a mistake by ordering the SPI version, and even though it is possible to reconfigure the interface of this module, I wouldn't recommend it for this project. Not only because it involves SMD soldering, but also because some additional components and wiring are needed.

Other than the Fritzing diagram (Figure 3) suggests, Aarav used a DHT11-module: a

board that does not only contains the DHT11 itself, but also (among others) a pull-up resistor at the data output pin. The internal pull-up resistor of an Arduino Nano input may be too high to ensure proper operation of the sensor, so an additional 10k pull-up resistor may be needed if you — like I did — use a separate DHT11 sensor. But with my own prototype, there was no need for it, the data signal looked nice and clean on an oscilloscope.

Even though building the hardware is relatively simple, I would advise to program the Arduino Nano and test the weather station first, before the modules are glued into the enclosure. It will be easier to correct possible soldering errors then.

the Arduino which in turn runs everything, our device is just a plastic box with no functionality. So let's program our pocket weather station for functionality and maximum efficiency. The Arduino sketch can be downloaded from this project's Elektor Labs page [1], but if you wish to, you can also get your hands dirty and write the code down yourself!

Step 11: And Here We Go!

And here we have our fully functional pocket weather station. It has an OLED display for you to enjoy the weather in a great manner. It also features a rechargeable battery with a USB charging port, and honestly, the battery lasts quite a long time so you would barely want to charge it. Also, it has an Arduino Nano port for uploading or tweaking the code in the future. The switch present on the outside is also very convenient to use. Besides this, the device's compact design helps it fit even in the smallest of pockets! Now, you can proudly flaunt the device wherever you go and actually use it as a weather station. ◀

210394-01

Contributors

Idea, Design and Text: **Aarav Garg**
Illustrations: **Aarav Garg,**
Patrick Wielders, Luc Lemmens
Editor: **Luc Lemmens**
Layout: **Giel Dols**

Questions or Comments?

Do you have any technical questions or comments about this article? Email the author at gargaarav79@gmail.com or contact Elektor at editor@elektor.com.



RELATED PRODUCTS

- **JOY-it Nano V3 (SKU 18615)**
www.elektor.com/18615
- **Blue 0.96" OLED Display I²C 4-pin (SKU 18747)**
www.elektor.com/18747
- **Book: *The Ultimate Compendium of Sensor Projects* (SKU 19103)**
www.elektor.com/19103



WEB LINKS

- [1] This project's Elektor Labs page: <https://www.elektormagazine.com/labs/pocket-weather-station>
- [2] Wikipedia on 'Heat Index': https://en.wikipedia.org/wiki/Heat_index

Lithium Battery Pack Repair

Save Money + More Power!

By **Dr. Thomas Scherer** (Germany)

It's a familiar story. After a few years of use, the cordless screwdriver needs charging more often and the cordless vacuum cleaner doesn't have the energy to collect your crumbs. In my case, it was the robot lawnmower giving up the ghost mid-mow, stubbornly refusing to return to its charging station. The simplest and most costly solution is to order a replacement battery pack. But have you considered just replacing the cells in the battery pack? This approach saves money and reduces waste. Furthermore, you can select replacement cells with a larger capacity than the originals. This isn't just a repair; it's an upgrade!

Battery-powered equipment running on Li-ion cells certainly retains its performance much longer compared to the NiMH cell-based power tools of the past. However, after many charge/discharge cycles, there comes a time when the energy storage capacity of even the best lithium battery drops so low that the battery pack needs to be replaced. I have already seen this with many of my own devices, and friends and colleagues often turn to me for advice on this topic.

The simplest solution is to visit the equipment manufacturer's website to see if a replacement battery pack is available. Sometimes there isn't and, when there is, the prices can come as a shock. In my case, the equipment worked just fine and looked to have a good few more years of life in it — a new battery would suffice. In such cases, it can be worthwhile hacking the battery pack and replacing the individual cells when the time comes, which is often cheaper overall. We can even consider improving the performance by replacing the original cells with some of a higher specification. If you also choose this route, you'll need to fire up the soldering iron in addition to breaking out the screwdrivers.

It's All Gone Quiet...

In my case, I noticed that my Robbi lawnmower [1] only wanted to mow for half an hour before it beetled off to its charging station for a 1.5-hour recharge. Previously, the usual pattern had been an hour of mowing followed by a 1-hour recharge. Was this change in routine a sign? Having had Robbi for four years and knowing it is powered by lithium cells, it was clear that they would likely need replacing.

Later that afternoon, I realized it had been quiet outside for some time. There, in the middle of the lawn, Robbi had shut down and couldn't be woken by pressing its buttons. I lugged it over to the charging station and put it on charge. Robbi eventually sprang into life, and I read from a menu option that its elapsed operating time was 2,938 hours. Corresponding to almost 1,500 charge cycles, it was clear that I should consider sourcing a new battery soon.

'Soon' came earlier than expected. After its full charge, Robbi finished its work, but it didn't wake up the next morning. Putting it on charge

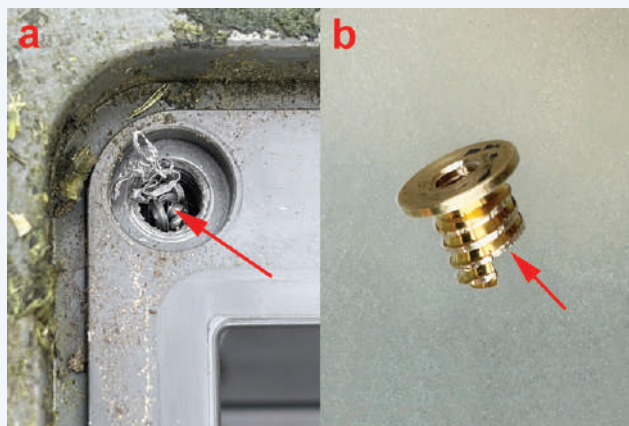


Figure 1: One of the screws had been cross-threaded during assembly. It eventually came out but took the captive nut with it. It was repaired using epoxy.

didn't help either. I had recently treated the lawn with some fertilizer, and we have had quite a lot of rain, so I needed to act quickly. I could almost hear the grass growing...

Hack a Pack?

The manufacturer's replacement battery pack was priced at around € 100, and a replacement from a third-party supplier was available for around half that price, which is not that bad. From its specification, I was looking for an 18 V replacement pack with a capacity of 2.1 Ah. That meant five cells, probably in the standard 18650 outline. To confirm my suspicions, I set about removing the lid of the battery compartment.

Unsurprisingly, this was easier said than done. Three of the screws came out easily, but the fourth was completely jammed. Eventually, it came out, but the captive mounting nut was ripped out in the process, jammed on the screw thread. It looked like it has been cross-threaded during assembly at the factory. **Figure 1a** shows the captive nut recess

in the mower casing after removal. Some debris remained. I eventually separated the screw and nut and, despite part of the captive brass nut breaking off (**Figure 1b**), it looked repairable.

The battery compartment is designed to be waterproof, so the captive nut required remounting to ensure all four screws could be tightened for a good seal. I used an epoxy adhesive to secure the nut, and you can see the result in **Figure 2b**. The brass nut at the top left looks as if it's been there forever. With the mechanical part fixed, attention turned to the battery pack. **Figure 2b** shows the battery in its compartment. The outline of five cells is clearly visible, and a ruler confirmed that they were 18650 cells. There is also a lot of free space (**Figure 2c**), which got me thinking: could I better use the space by using more or larger replacement cells? My mind was now made up; purchase of an off-the-shelf replacement battery was no longer an option.

Cell Swap

Figure 2a shows that two recesses in the battery lid encroach into the available battery space, ruling out the fitting of two rows of five cells to double capacity. There are, however, more expensive cells in the 18650 format with higher capacity. Some of the better-known brands have cells in this format with ratings of up to 3,500 mAh. Such high-capacity branded cells cost a good 10 € each. You can also find cells with much higher ratings on eBay, AliExpress, and similar sites, but you should take such claims with a pinch of salt.

I almost placed an order for five cells until I noticed some others in the somewhat more unusual 21700 format. Although only slightly larger, they offered significantly more capacity for the money. I was sure they would fit in somehow (I was thinking of a row of three and a row of two to form a W profile). With a quantity discount and postage, the 5 × 4,000 mAh cells came to € 26 in total. They dropped through the letterbox two days later, and it wasn't a day too soon — the grass was now definitely in need of a trim.

Considerations

Building a battery pack from individual cells generally requires a degree of dexterity, electrical expertise, and a spot welder. As you can see

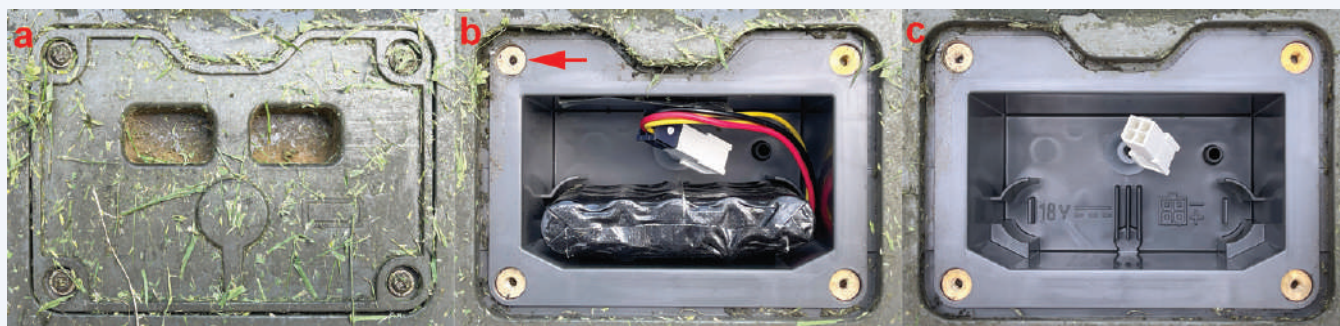


Figure 2: Battery compartment lid (2a), with the old battery in situ (2b) and the empty compartment (2c).

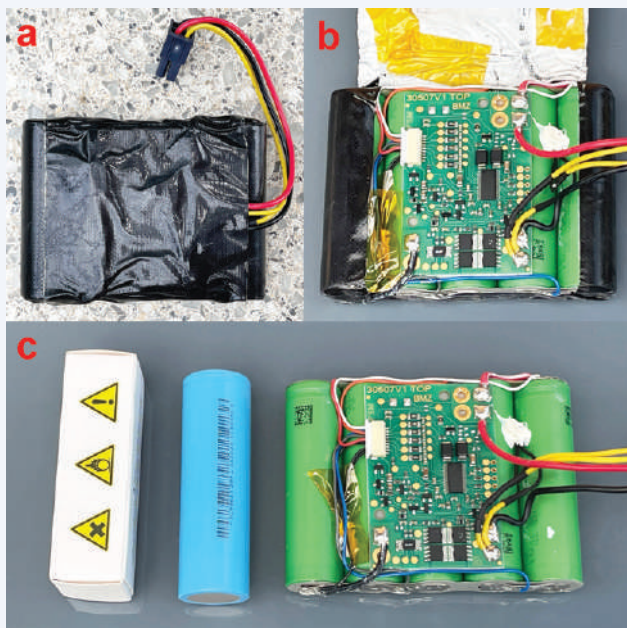


Figure 3: The old battery pack is wrapped in black gaffer tape (3a). The cells and BMS board are visible with the top layer of tape removed (3b). A boxed and unboxed 21700 cell in comparison with the old 5-cell 18650 pack (3c).

from the old unwrapped battery pack in **Figure 3**, the five green cells are neatly connected at the positive and negative contacts via thin, spot-welded nickel strips. This type of permanent connection method does not damage the cell. Although the heat generated during spot welding is intense, it is swift and localized, so the cells are barely warmed. It's well known you can never have too many tools, but, in this case, I can't imagine where else I would use a spot welder, so I can't justify the outlay. Instead, I chose to solder the wires directly to the cell contacts. This is not recommended unless you follow some rules and are aware of the dangers.

The battery pack used in **Figure 3** is typical of that found in many other battery-operated devices. It consists of several battery cells connected in series plus a BMS (Battery Management System) PCB. This is the circuit board shown in **Figures 3b** and **3c**. The latter image also shows a size comparison between the new cells and those in the old battery pack. The BMS performs three basic functions:

BMS Flip-Flop Function

When I removed the battery shown in Figure 3c, I was surprised to measure 19.2 V directly on the plus and minus contacts of the 5-cell battery pack. Had I been too hasty in ordering replacements? On the other hand, I could only measure about 18.5 V at the battery pack connector where it connects to the mower motor (to the right of the BMS board), which had a very high source impedance. Using just my fingers, I could discharge this to ground, causing the voltage to drop to just a few volts. Was the BMS broken?

I connected a 24 Ω load resistor directly to the cell contacts on the battery pack and measured a current of 0.75 A and a battery voltage of 18.2 V. I then disconnected the load and put the pack onto charge. After just a few seconds of charging at 0.5 A, the voltage output from the BMS switched to low impedance mode to draw some current from the battery via the BMS. It looked like the BMS had detected the battery voltage falling below the undervoltage threshold when the mower was last used and had turned off the FET to disconnect the battery. This 'off' state had been latched in the BMS. To test it, I connected a 12 Ω load to the battery. After five minutes, the BMS switched off at just over 13 V and switched on again after another charge.

Phew... With the BMS good to go, I proceeded with the cell swap.

1. It balances the cells, i.e., keeps them all at the same voltage or charge state.
2. It prevents the cells from being overcharged.
3. It disconnects the load in the event of undervoltage to avoid deep discharge.

The chip on the BMS board with the most legs manages all these tasks. It is a specialized microcontroller that monitors the cell voltages (via the connector on the left). In the event of an overvoltage or undervoltage condition, it disconnects the cells using 2×2 MOSFETs (at the bottom). Further reading on balancing lithium batteries is available under [2] and [3].

The BMS is included if you buy a whole new battery pack assembly, so the BMS board (which still works) from the old pack will be redundant. On the other hand, if you only swap the cells, you can reuse the existing BMS board. An important feature of the BMS that you need to be aware of is described in the text box **BMS Flip-Flop Function**.

WEB LINKS

- [1] T. Scherer, "Solar Power for Mowing Robots," *Elektor*, June/July 2021: <https://bit.ly/3AgktGu>
- [2] T. Scherer, "LiPo Auto Balancer," *Elektor*, June 2010: <https://bit.ly/3AhZa7e>
- [3] "Battery Management System Tutorial," *Elektor Business Magazine*, February 2017: <https://bit.ly/3ly9cND>

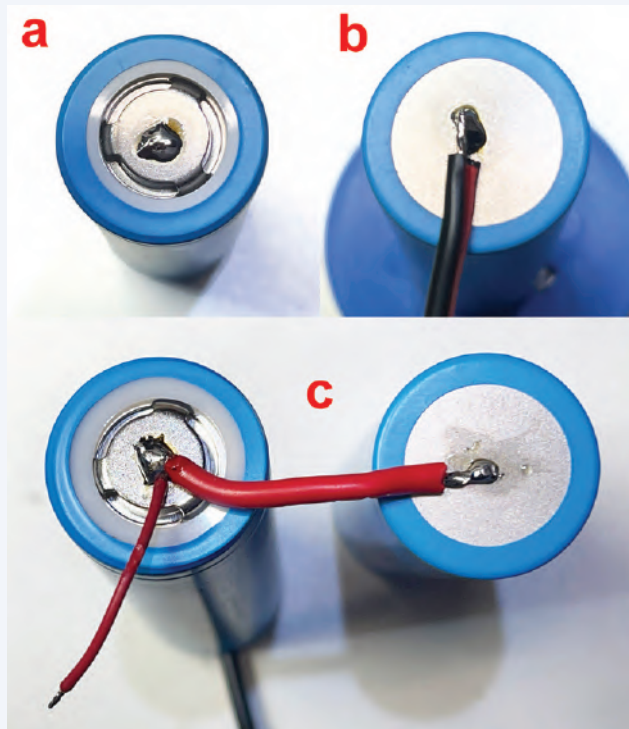


Figure 4: The positive contact is more tolerant of the soldering procedure (4a). Short lengths of wire are soldered to the negative contact (4b). The cells are finally connected in series, and the thinner voltage sense wires are connected to the BMS (4c).

Keep Soldering On

With no spot welder to hand, I decided to solder stranded wire directly to the battery terminals. As long as you are careful, this can be done without harming the batteries. Any thermal damage inflicted on the constituent materials of the cells is roughly proportional to the integral of time and temperature. In other words, you need to be quick! Three things are important here. Firstly, the soldering iron must be powerful enough for the tip to maintain its temperature during soldering. This ensures that the end of the cell quickly achieves soldering temperature. A solder that melts at a lower temperature is also advisable. I used a 90 W iron as it has a regulator that allows the bit temperature to be set at over 400 °C. It is best to avoid lead-free solder for this job as it melts at a higher temperature and does not wet the surface as nicely as the good old SnPb 60/40 that I prefer. In my experience, the metal contact surfaces of the lithium cells take the solder easily. With the tip temperature set to 385 °C and using 1 mm diameter, flux-cored solder, I completed each joint in around one second - fast enough to avoid damaging the cell.

If you need to keep the soldering tip on the cell contacts for much longer than this (because the soldering iron is underpowered, the temperature is too low, or through the use of lead-free solder), you risk overheating and damaging the cell. This will impact the cell's electrical capacity and possibly reduce the number of charge/discharge cycles. As long as the iron is only in contact with the surface for around a second, it shouldn't cause any harm. Alternatively, you could buy some slightly more expensive tagged cells that come with a short nickel strip already spot welded onto the cell contacts. The strips can then be soldered

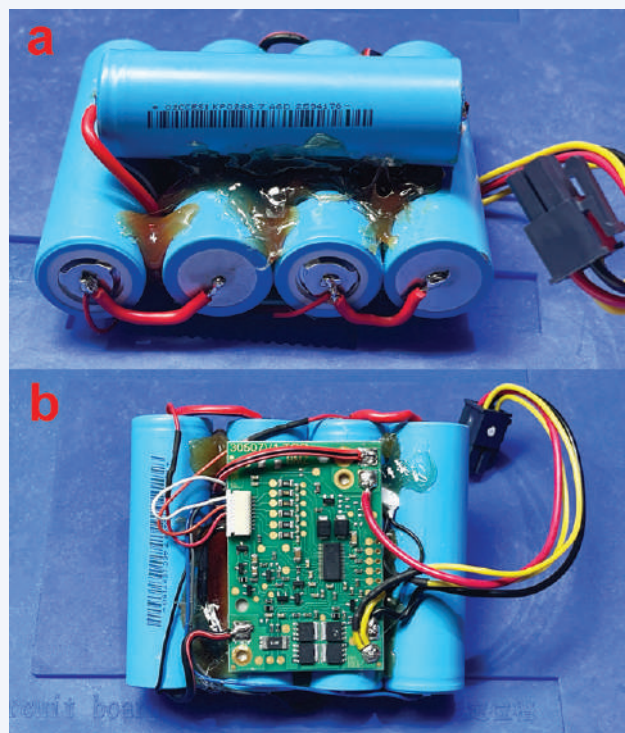


Figure 5: Underside of the finished pack (5a) and top view with the BMS board (5b).

together to make up the pack without any risk of overheating the cell contents. In principle, this is safer, but you must insulate the bare strips to prevent any short circuit. A short circuit during normal operation will create a significantly more dangerous condition than could occur by applying a short blast of heat to solder wires to the cell contacts.

A close-up of my soldering method can be seen in **Figure 4**. The positive contact consists of a metal cap attached to the battery electrode at three points in this cell. This arrangement increases the thermal resistance between the outside contact surface and the cell's internal structure, making it more tolerant of the soldering process. Start by applying small drops of solder to tin all the positive contacts (**Figure 4a**) quickly. Next, cut 1.5 mm² stranded wire to a length of 3 cm, which allows some leeway for later positioning of the cells in the pack. These can now be soldered directly onto the negative contacts (**Figure 4b**) in two stages. Firstly, use the soldering iron and solder them quickly to establish a small tinned area in the center of the contacts. Next, strip and tin the ends of the connecting wires. Once the cells have cooled, quickly solder a wire to each of the negative contacts. In **Figure 4c**, you can see the other end of a wire now soldered to the positive contact of the next cell to make the series connection. The thinner red wire is the cell voltage sense connection to the BMS board.

Assemble and Test

As already mentioned, the battery compartment cannot accommodate the five cells arranged in rows of two and three to form a W configuration, so I had to find a different pack construction. In **Figure 5**, you can see that four of the cells are placed next to one other and fixed in

place with hot glue. The fifth cell is glued across the other four. The hot glue makes the finished assembly very stable. You could alternatively use silicone sealant here.

The double-sided tape on the back of the BMS board did not need to be replaced as it held firmly to the new battery pack. All that remained was to connect the six wires from the white connector (**Figure 5b**) to the corresponding cell contacts, along with the plus and minus connections of the entire pack, to the BMS board. This must be completed before we can test the new battery. Double-check all the wiring to ensure you haven't made any mistakes. Taking a photo of the old battery pack will help at this later stage when checking everything. The battery pack functioned as expected and could be charged and discharged without a problem.

The finished battery pack was wrapped in gaffer tape to make the assembly more robust and provide insulation and moisture ingress protection (**Figure 6**) before fitting into the mower's battery compartment. After screwing down the lid and switching on the mower, it wanted to calibrate itself with the signal from the guide cable and to start moving immediately. I canceled the latter operation and put it into its charging station. It needed a full three hours to charge, indicating that the battery now has almost twice the capacity of the original.

The robot mows as well as it ever has — an hour of mowing followed by an hour of charging. The battery is only partially discharged during these cycles, so I assume that these larger capacity cells will tolerate significantly more charging cycles before they need replacing again. With any luck, I reckon the new pack should last twice as long as the original. If that is the case, the effort involved in three hours of tinkering was probably worthwhile. Working out the hourly wage rate for the time taken, the fiscal aspect looks poor, but, on the plus side, I now have a solution that can't be bought off the shelf.

This method of battery cell replacement isn't limited to lawnmowers. You can use the same approach to extend the life of cordless vacuum cleaners and other devices, even if the available battery space is less generous. The last vacuum cleaner I repaired using the method (using high-capacity 18650 cells) has been in use for three years without any performance complaints. ◀

210368-01

Questions or Comments?

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

Contributors

Idea, Design and Text: **Dr. Thomas Scherer**
Editor: **Jens Nickel, Stuart Cording**
Translation: **Martin Cooke**
Layout: **Giel Dols**

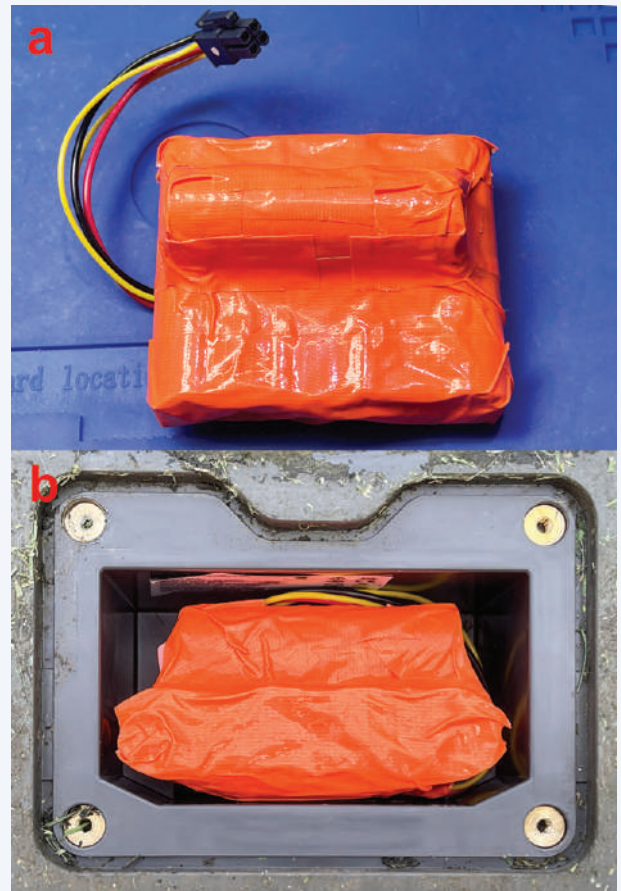


Figure 6: The new pack wrapped in orange gaffer tape (6a) fits neatly in the housing (6b).



RELATED PRODUCTS

- > **OWON OW16B Digital Multimeter with Bluetooth (SKU 18780)**
www.elektor.com/18780
- > **PeakTech 4350 Clamp Meter (SKU 18161)**
www.elektor.com/18161
- > **JOY-it RD6006 DC Power Supply (360 W) (SKU 19564)**
www.elektor.com/19564

Create GUIs with Python: Meme Generator

Create a GUI application which draws memes



Laura Sach

Laura leads the A Level team at the Raspberry Pi Foundation, creating resources for students to learn about Computer Science.

@CodeBoom

Let's take the lessons you learnt from the previous instalments to create a GUI which draws memes. You will input the text and image name and your GUI will combine them into your own meme using the Drawing widget.

Start by creating a simple GUI with two text boxes for the top and bottom text. This is where you will enter the text which will be inserted over your picture to create your meme. Add this line to import the widgets needed.

```
from guizero import App, TextBox, Drawing
```

Then add this code for the app:

```
app = App("meme")

top_text = TextBox(app, "top text")
bottom_text = TextBox(app, "bottom text")

app.display()
```

The meme will be created on a Drawing widget which will hold the image and text.

Create a meme

Add it to the GUI by inserting this code just before the `app.display()` line. The Drawing widget's height and width should be set to 'fill' the rest of the GUI.

```
meme = Drawing(app, width="fill",
height="fill")
```

The meme will be created when the text in the top

and bottom text boxes changes. To do that, we will need to create a function which draws the meme.

The function should clear the drawing, create an image (we're using a photo of a woodpecker, but you can use any you want) and insert the text at the top and bottom of the image.

Remember when you used `name.value` to set the value of the Text widget with the spy name in Part 2 of this series? You can also use the value property to get the value of a Text widget, so in this case `top_text.value` means 'please get the value that is typed in the top_text box'.

```
def draw_meme():
    meme.clear()
    meme.image(0, 0, "woodpecker.png")
    meme.text(20, 20, top_text.value)
    meme.text(20, 320, bottom_text.value)
```

The first two numbers in `meme.image(0, 0)` and `meme.text(20, 20)` are the x, y co-ordinates of where to draw the image and text. The image is drawn at position `0, 0`, which is the top-left corner, so the image covers the whole of the drawing.

Finally, call your `draw_meme` function just before you display the app. Insert this code just before the `app.display` line:

```
draw_meme()
```

Your code should now look like `meme1.py`.

If you run your app (**Figure 1**) and try changing the top and bottom text, you will notice that it doesn't update in the meme. To get this working, you will have to change your program to call the

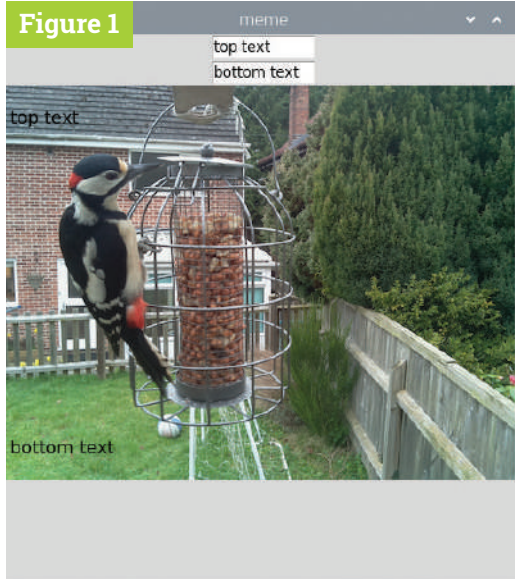


Martin O'Hanlon

Martin works in the learning team at the Raspberry Pi Foundation, where he creates online courses, projects, and learning resources.

@martinohanlon

Figure 1



▲ Figure 1 Meme with unstyled text

`draw_meme` function when the text changes, by adding a command to the two `TextBox` widgets to the app.

```
top_text = TextBox(app, "top text",
command=draw_meme)
bottom_text = TextBox(app, "bottom text",
command=draw_meme)
```

Your code should now look like that in **meme2.py**. Run it and update your meme by changing the top and bottom text.

“ Update your meme by changing the top and bottom text ”

You can alter the look of your meme by changing the `color`, `size`, and `font` parameters of the text. For example:

```
meme.text(
    20, 20, top_text.value,
    color="orange",
    size=40,
    font="courier")
meme.text(
    20, 320, bottom_text.value,
    color="blue",
    size=28,
    font="times new roman",
)
```

meme1.py

► Language: Python 3

DOWNLOAD
THE FULL CODE:



magpi.cc/guizero/code

```
001. # Imports -----
002.
003. from guizero import App, TextBox, Drawing
004.
005.
006. # Functions -----
007.
008. def draw_meme():
009.     meme.clear()
010.     meme.image(0, 0, "woodpecker.png")
011.     meme.text(20, 20, top_text.value)
012.     meme.text(20, 320, bottom_text.value)
013.
014.
015. # App -----
016.
017. app = App("meme")
018.
019. top_text = TextBox(app, "top text")
020. bottom_text = TextBox(app, "bottom text")
021.
022. meme = Drawing(app, width="fill", height="fill")
023.
024. draw_meme()
025.
026. app.display()
```

meme2.py

► Language: Python 3

```
001. # Imports -----
002.
003. from guizero import App, TextBox, Drawing
004.
005.
006. # Functions -----
007.
008. def draw_meme():
009.     meme.clear()
010.     meme.image(0, 0, "woodpecker.png")
011.     meme.text(20, 20, top_text.value)
012.     meme.text(20, 320, bottom_text.value)
013.
014.
015. # App -----
016.
017. app = App("meme")
018.
019. top_text = TextBox(app, "top text", command=draw_meme)
020. bottom_text = TextBox(app, "bottom text", command=draw_meme)
021.
022. meme = Drawing(app, width="fill", height="fill")
023.
024. draw_meme()
025.
026. app.display()
```

Top Tip



These lines of code were starting to get very long, so we have split them over a number of lines to make it easier to read. It doesn't affect what the program does, just how it looks.

Your code should now look like **meme3.py**. Try different styles until you find something you like (**Figure 2**).

Customise your meme generator

For a truly interactive meme generator, the user should be able to set the font, size, and colour themselves. You can provide additional widgets on the GUI to allow them to do this.

The number of options available for the colour and font are limited, so you could use a drop-down list, also known as a Combo, for this. The size could be set using a Slider widget.

First, modify your import statement to include

the Combo and Slider widgets.

```
from guizero import App, TextBox, Drawing,
Combo, Slider
```

After you have created your TextBox widgets for the top and bottom text, create a new Combo widget so the user can select a colour.

```
bottom_text = TextBox(app, "bottom text",
command=draw_meme)
color = Combo(app,
options=["black", "white", "red",
"green", "blue", "orange"],
command=draw_meme)
```

The **options** parameter sets what colours the user can select from the Combo. Each colour is an element in a list. You can add any other colours you want to the list.

The options are displayed in the order in which you put them in the list. The first option is the default, which is displayed first. If you want to have a different option as the default, you can do it using the **selected** parameter, e.g. **"blue"**.

```
color = Combo(app,
```

“ The options are displayed in the order in which you put them in the list ”

```
options=["black", "white", "red",
"green", "blue", "orange"],
command=draw_meme,
selected="blue")
```

Now your user can select a colour. Next, you need to change the **draw_meme** function to use Combo's value when creating the text in your the meme. For example:

```
meme.text(
20, 20, top_text.value,
color=color.value,
size=40,
font="courier")
```

meme3.py

► Language: **Python 3**

```
001. # Imports -----
002.
003. from guizero import App, TextBox, Drawing
004.
005.
006. # Functions -----
007.
008. def draw_meme():
009.     meme.clear()
010.     meme.image(0, 0, "woodpecker.png")
011.     meme.text(
012.         20, 20, top_text.value,
013.         color="orange",
014.         size=40,
015.         font="courier")
016.     meme.text(
017.         20, 320, bottom_text.value,
018.         color="blue",
019.         size=28,
020.         font="times new roman",
021.     )
022.
023.
024. # App -----
025.
026. app = App("meme")
027.
028. top_text = TextBox(app, "top text", command=draw_meme)
029. bottom_text = TextBox(app, "bottom text", command=draw_meme)
030.
031. meme = Drawing(app, width="fill", height="fill")
032.
033. draw_meme()
034.
035. app.display()
```




▲ Figure 2 Alter the fonts and colours

Do the same for the bottom-text block of code. Your program should now resemble **meme4.py**.

Following the steps above, add a second Combo to your application so the user can select a font from this list of options: ["times new roman", "verdana", "courier", "impact"]. Remember to change the `draw_meme` function to use the `font` value when adding the text.

Create a new Slider widget to set the size of the text your user wants.

```
size = Slider(app, start=20, end=40,
              command=draw_meme)
```

The range of the slider is set using the `start` and `end` parameters. So, in this example, the smallest text available will be 20 and the largest 40.

Modify the `draw_meme` function to use the value from your size slider when creating the meme's text.

```
meme.text(
    20, 20, top_text.value,
    color=color.value,
    size=size.value,
    font=font.value)
```

meme4.py

> Language: **Python 3**

```
001. # Imports -----
002.
003. from guizero import App, TextBox, Drawing, Combo, Slider
004.
005.
006. # Functions -----
007.
008. def draw_meme():
009.     meme.clear()
010.     meme.image(0, 0, "woodpecker.png")
011.     meme.text(
012.         20, 20, top_text.value,
013.         color=color.value,
014.         size=40,
015.         font="courier")
016.     meme.text(
017.         20, 320, bottom_text.value,
018.         color=color.value,
019.         size=28,
020.         font="times new roman",
021.     )
022.
023.
024. # App -----
025.
026. app = App("meme")
027.
028. top_text = TextBox(app, "top text", command=draw_meme)
029. bottom_text = TextBox(app, "bottom text", command=draw_meme)
030.
031. color = Combo(app,
032.               options=["black", "white", "red", "green",
033.                       "blue", "orange"],
034.               command=draw_meme, selected="blue")
035.
036. meme = Drawing(app, width="fill", height="fill")
037.
038. draw_meme()
039. app.display()
```

Drawing widget

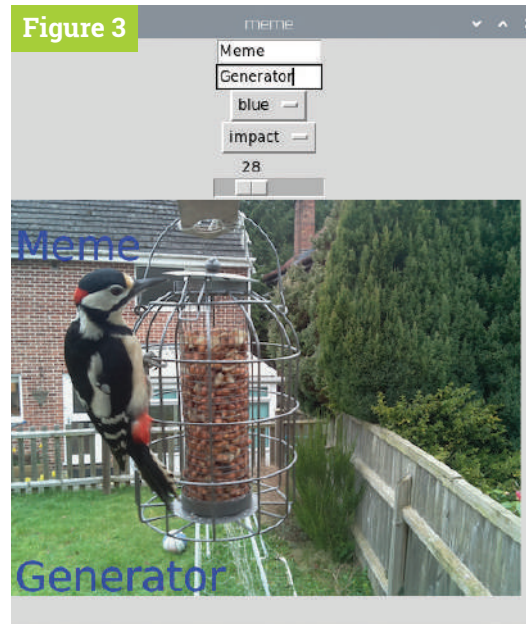
The Drawing widget is really versatile and can be used to display lots of different shapes, patterns, and images. To find out more about the Drawing widget, see Appendix C of the book (magpi.cc/pythongui), or take a look at the online documentation: lawsie.github.io/guizero/drawing.

04-meme-generator.py

► Language: **Python 3**

```
001. # Imports -----
002.
003. from guizero import App, TextBox, Drawing, Combo, Slider
004.
005.
006. # Functions -----
007.
008. def draw_meme():
009.     meme.clear()
010.     meme.image(0, 0, "woodpecker.png")
011.     meme.text(
012.         20, 20, top_text.value,
013.         color=color.value,
014.         size=size.value,
015.         font=font.value)
016.     meme.text(
017.         20, 320, bottom_text.value,
018.         color=color.value,
019.         size=size.value,
020.         font=font.value,
021.     )
022.
023.
024. # App -----
025.
026. app = App("meme")
027.
028. top_text = TextBox(app, "top text", command=draw_meme)
029. bottom_text = TextBox(app, "bottom text", command=draw_meme)
030.
031. color = Combo(app,
032.               options=["black", "white", "red", "green",
033.                       "blue", "orange"],
034.               command=draw_meme, selected="blue")
035.
036. font = Combo(app,
037.              options=["times new roman", "verdana", "courier",
038.                      "impact"],
039.              command=draw_meme)
040.
041. size = Slider(app, start=20, end=50, command=draw_meme)
042.
043. meme = Drawing(app, width="fill", height="fill")
044. draw_meme()
045. app.display()
```

Figure 3



▲ Figure 3 The finished meme generator

Your code should now resemble that in **04-meme-generator.py**. Try running it and you should see something like **Figure 3**.

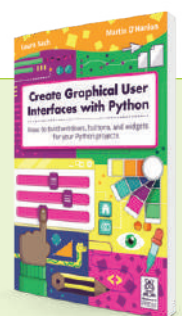
Can you change the GUI so that the name of the image file can be entered into a TextBox or perhaps selected from a list in a Combo? This would make your application capable of generating memes with different images too. [\[M\]](#)

Editor's note: This article originally appeared in *MagPi* magazine. Elektor publishes the Dutch, French, and German editions of *MagPi*.

Create Graphical User Interfaces with Python

For further tutorials on how to make your own GUIs with guizero, take a look at our book, *Create Graphical User Interfaces with Python*. Its 156 pages are packed with essential info and a range of exciting projects.

magpi.cc/pythongui



The official Raspberry Pi magazine



£10
for 3 issues

You also receive
a **FREE** book of
your choice

From an approved selection

SUBSCRIBE magpi.cc/freebook **CALL** 01293 312193

The investment program



We connect start-ups with future customers, partners and resellers. Learn about the benefits for both start-ups and investors.

www.elektormagazine.com/investment-program



 **elektor**
design > share > sell

Three Questions to Build On

Why
What
Who

By **Priscilla Haring-Kuipers**
(The Netherlands)

The considerations might be very different when you are building a health-related device, a musical instrument, or a fabulous IoT object. But I think the questions are the same.

Why are you building this? Does it have sufficient purpose to excuse its existence? What would happen if this thing was not made (by you)?

What are the materials and processes involved in making it? What happens to the material during its lifetime and at the end of it?

Who are the people involved in the process of making this and how are they treated? What does your thing mean to the humans that will end up having it? What does making it mean to you?

Let me dive into these questions as a boutique synthesizer manufacturer.

Why

I think anyone building anything has a little (or a lot) of the artistic need to create. Often the Things we end up producing start from a *need* to create this one particular

sequencer that is nagging to be made, or a different approach to a low-frequency oscillator that is clearly missing in the world. Answering this need in itself has value *if* you think art and creativity has value. For me this need alone would be reason enough to make one, but not reason enough to produce 100. Once something has pushed itself forward out of this *need*, we ask if it will add anything to the world. We make our own very subjective judgments by looking at what is available in the (modular) synthesizer market and hypothesizing if our Thing is different enough where it matters. We are also asking ourselves if we have the correct skills to make our Thing achieve its intended purpose. Since we are the only ones who are concerned that this thing must be made in the first place, if we can technically build it to make this purpose happen, then the answer usually is “yes.”

What

Then we start considering What it would entail to build this Thing. We think about the materials and processes involved and consider our choices. Here we also determine — almost imperceptibly — what level of choice we are morally willing to accept. We are aware that we are not doing absolutely everything we can. We have made no ethical choices concerning the raw materials or components. We would like more informed choices, but we accept and work with what is easily available to us.

Our first answer to the sustainability question is to build something that is meant to last, and this guides any material choices. We expect our Things to have 20-100 years of life, so everything inside and outside must be able to live that long or be replaced. This is also inherent to the type of electronics we make. Musical instruments are loved, maintained, and repaired. We facilitate this by sharing our designs and supporting customers and music stores to fix things themselves if they can or sent any Thing back to us for repairs.

We have chosen to use both aluminium and bamboo wherever we can. Aluminium is already very recycled. Our front plate factory uses an about 50% recycled aluminium mix, which is pretty standard in the industry and can be recycled again. Bamboo has the properties of a hard wood without having to chop down forests for it. Both materials are super durable and aesthetically pleasing.

The recycling of our Things has not been properly considered as we have not been around long enough for this to be relevant. Waste from our prototyping goes to the relevant local recycling stations.

Who

There are humans involved everywhere in the making of our Things — including ourselves. The ones that we know the least about are the farthest away from us in the chain. We don't know who is mining our raw materials or who is making them into the components we use. Some components

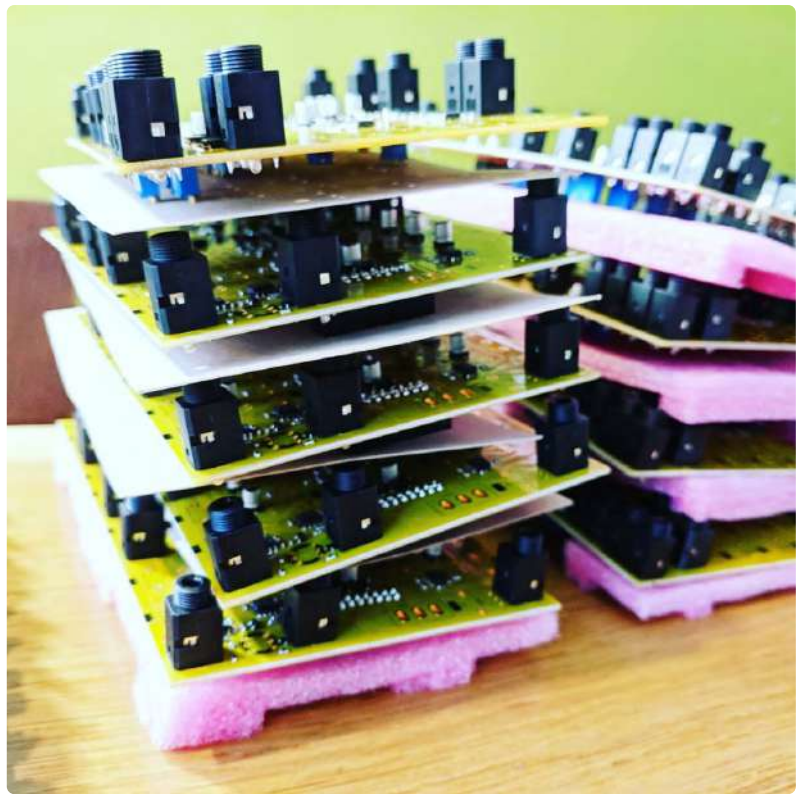


Figure 1: Boards that are inside the RectangularThing modules.

we buy straight from factories in Taiwan, and we assume this means that they are made there, but we really have no idea. Our PCBs come from Shenzhen where we have visited several PCB factories and assembly houses (one of which we suspect to have been a show factory). Labour circumstances are improving, but they are still nowhere near what we would find acceptable in the European Union. We decided to have most of our assembly done in the Netherlands, in part because the working conditions here are what we consider ethical. We have visited these factories too and have had tea with the women building our electronics.

We speculate how other people would use our Thing. We believe that creating sounds, making music, performing music, listening, and dancing to music are all worthy pursuits that make the world a better place. We are gratified when we feel that we will

contribute to this with our Things, and we aim to facilitate creativity in music making wherever we can. We design for this. We want to engage with the users of our Things in ways that enriches our lives as well.

This whole puzzle has to end up being economically feasible. If we make this Thing that we are sure we want to make, with the materials we want to use, and involving the people we want to make it with, is it likely that people will want to buy our Thing at the price we arrive at? Only if the answer to all of these things is “Yes” do we start a production run, usually of 100 units.

Have you asked yourself Why, What, and Who when you are about to build something? Please share your considerations. ◀

210663-01

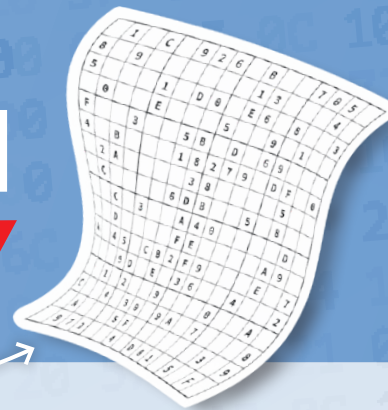


The World Ethical Electronics Forum

The purpose of the World Ethical Electronics Forum (WEEF) is to inspire global innovators in electronics with an open discussion about ethics and sustainable development goals. The inaugural WEEF event took place on November 18, 2021 in Munich, Germany. Visit the WEEF webpage — www.elektormagazine.com/weef — for details about the event and for future news updates about WEEF 2022.

Hexadoku

Puzzles with an Electronic Touch



Traditionally, the last page of *Elektor* magazine is reserved for our puzzle with an electronics slant: welcome to Hexadoku! Find the solution in the gray boxes, submit it to us by email, and you automatically enter the prize draw for one of five Elektor store vouchers.

The Hexadoku puzzle employs 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 store vouchers worth **€50.00 each**, which should encourage all Elektor readers to participate.

PARTICIPATE!

Ultimately April 15th, 2022, 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 01-02/2022 (January & February) is: **FA028**.

Solutions submitted to us before February 15th were entered in a prize draw for 5 Elektor Store Vouchers.

The winners are posted at www.elektormagazine.com/hexadoku.

Congratulations everyone!

E	2	6			3	C			5	7			4		D
4						5		E			6	A			
				E	F			C		8	A	3	6		9
		1	A	2		7			D				C		E
A				5			8	4	6		9		7		
	D	B						7		5	C				
	5				A	B		3					0	D	
				7		0	1		B		2				
				A		F	B		1		7				
	6				8	E		4					D	0	
	4	5						A		E	6				
B				7			4	9	0		D		1		
		3	2	8		6			E				9		B
				F	E			7		5	8	D	2		3
C					2		A			0	8				
D	9	8			1	B			F	2			E		4

2	4	7	E	3	D	B	F	A	5	C	0	9	6	8	1
8	1	D	F	C	9	0	2	B	7	E	6	3	A	4	5
B	0	9	5	A	4	1	6	D	F	3	8	7	C	E	2
C	A	3	6	5	7	8	E	9	1	2	4	F	B	D	0
3	F	0	7	1	8	2	C	E	A	5	9	B	4	6	D
9	2	1	A	4	3	7	5	6	C	B	D	8	E	0	F
4	5	8	B	6	E	9	D	F	0	1	7	A	2	3	C
6	D	E	C	B	F	A	0	2	8	4	3	5	7	1	9
E	6	B	3	7	A	F	9	C	4	0	1	D	5	2	8
A	7	5	0	D	C	E	1	3	2	8	B	6	F	9	4
D	9	F	2	8	0	3	4	5	E	6	A	C	1	B	7
1	8	C	4	2	5	6	B	7	9	D	F	E	0	A	3
5	3	2	D	9	B	4	A	0	6	7	C	1	8	F	E
F	B	A	1	0	2	5	7	8	3	9	E	4	D	C	6
0	C	4	9	E	6	D	8	1	B	F	5	2	3	7	A
7	E	6	8	F	1	C	3	4	D	A	2	0	9	5	B

The competition is not open to employees of Elektor International Media, its subsidiaries, licensees and/or associated publishing houses.

Development tools in one location

Thousands of tools from hundreds
of trusted manufacturers



Choose from our
extensive selection at
mouser.com/dev-tools



MOUSER
ELECTRONICS



PROTEUS DESIGN SUITE

Design Quality Assurance

Constraint Driven Design

Flexible and scalable
rule system

Full support for design
rule rooms

Manufacturing
solder mask rules

Live display of
violation areas

Zone Inspector

Analyze plane coverage and
stitching

Grid view of plane
configurations

Edit plane settings and
draw order

Pre-Production Checklist

Set of board tests
before Gerber Output

Includes placement,
connectivity and
clearance testing

Completely independent
code for clearance checks

Dedicated Reporting Module

Tables automatically
populate with design
data

Compliance status for
diff pairs and length
matched routes

Make custom
reports with data
object tables

Generate reports
from templates

