£9.95 | €10.95 | $12.95

elektor MAG

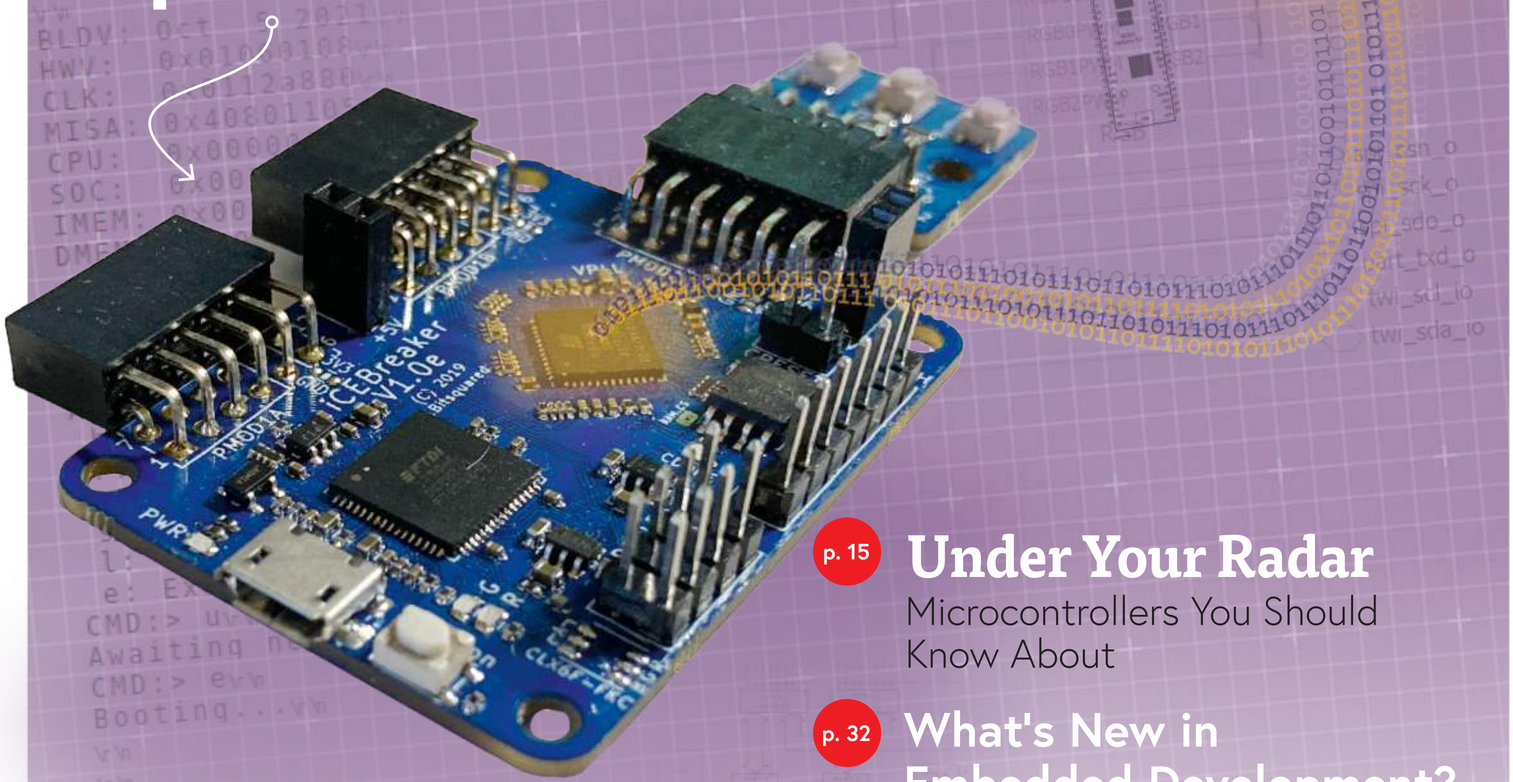
design > share > earn

embedded world Special * SINCE 1961 *

JUNE & JULY 2022
ELEKTORMAGAZINE.COM

embedded world Special

RISC-V: Build Your Own Open-Source Processor



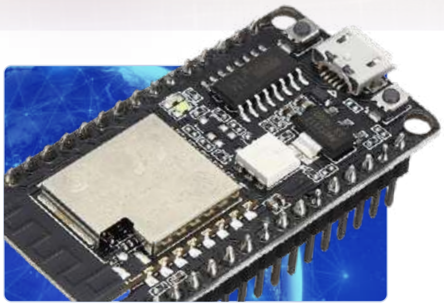
p. 15 **Under Your Radar**
Microcontrollers You Should Know About

p. 32 **What's New in Embedded Development?**
Rust and Keeping IoT Deployments Updated

FOCUS ON Embedded Development and IoT

embeddedworld2022
Exhibition&Conference

The Challenges in Bringing IoT Solutions to Market
Worries Around Security, Scalability, and Competition p. 70



First Steps with an ESP32-C3 and the IoT
A Wi-Fi Button and Relay p. 42

Preferably Wired After All
Developing a 1 Gbit/s Interface in an Industrial Environment p. 78

Elektor Magazine,
English edition

embedded world Special 2022
June & July 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

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

Memberships:
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.

Jens Nickel

International Editor-in-Chief, Elektor Magazine

No Substitute for an In-Person Fair

After two years of drastic pandemic measures, things are finally moving forward with in-person trade shows. At the first events, such as PCIM and Sensor+Test, it became clear that the whole thing was starting up again rather gently. No one could complain about crowding in the halls. I also heard at many booths that companies want to focus their trade show program more in the future: on one, at most two, trade shows per year.

For Elektor and many of our partner companies — which often come from the semiconductor, software and distribution sectors — **embedded world** (alongside electronica) is right at the top of the list. And so this trade show could become a litmus test of how much value companies still attach to a trade show appearance and visit, which always costs a lot of time and money.

We have made up our minds on this issue — just like the companies represented in this special. There is no substitute for a trade fair! Because only here can you get an overview of the market in a short time, discover ingenious new solutions, look around for sources of supply and customers, spot trends, and capture moods.

My colleagues and I will be doing just that here in Nuremberg, with lots of video interviews and a newsroom at our booth. I would like to invite you to visit us. We can be found in Hall 4A (4A.646)!

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

FIPP Connecting Global Media

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."

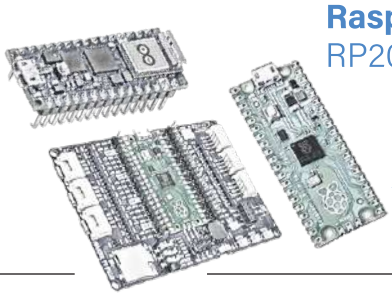
DEUTSCHE FACHPRESSE

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

THIS EDITION

Volume 48, embedded world Special
June & July 2022

Regulars & Features



The WinUI Graphics Framework for Windows Apps

Build Your Own RISC-V Controller

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

Projects

Next Edition

Elektor Magazine Edition 7-8/2022 (July & August 2022)
As usual, we'll have an exciting mix of projects, circuits, fundamentals and tips and tricks for electronics engineers and makers. We will focus on Test & Measurement.

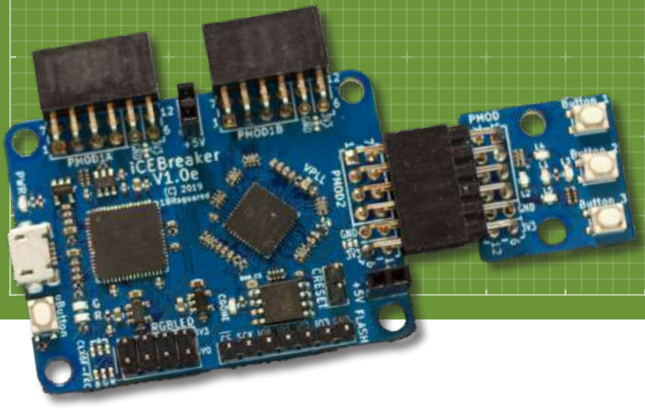
From the contents:
- Autonomous Inductance Meter
- CO_2 Meter with Sigfox
- Smart Plugs: A Look Inside and Hacked
- Simple Analog ESR Meter
- Get Started With Your Oscilloscope
- Raspberry Pi Pico Makes an MSF-SDR
- AC Grid Frequency Meter

And much more!

Elektor Magazine edition 7-8/2022 covering July & August 2022 will be published around July 7, 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 there 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 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.

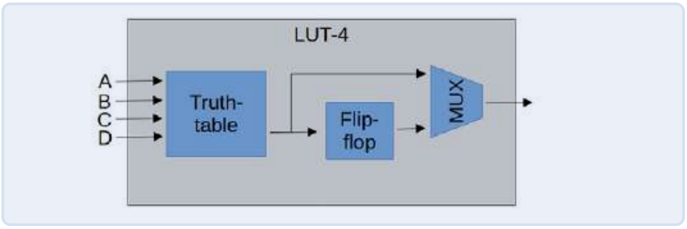


Figure 1: Signal flow in a LUT-4.

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, I^{2}C 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?

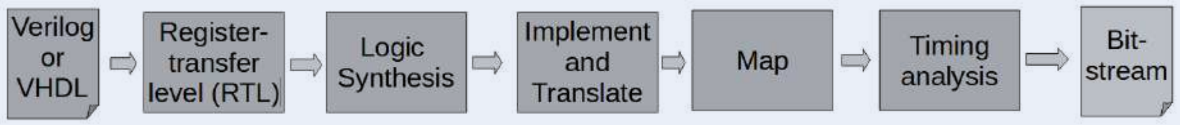


Figure 2: Steps in the FPGA synthesis process.

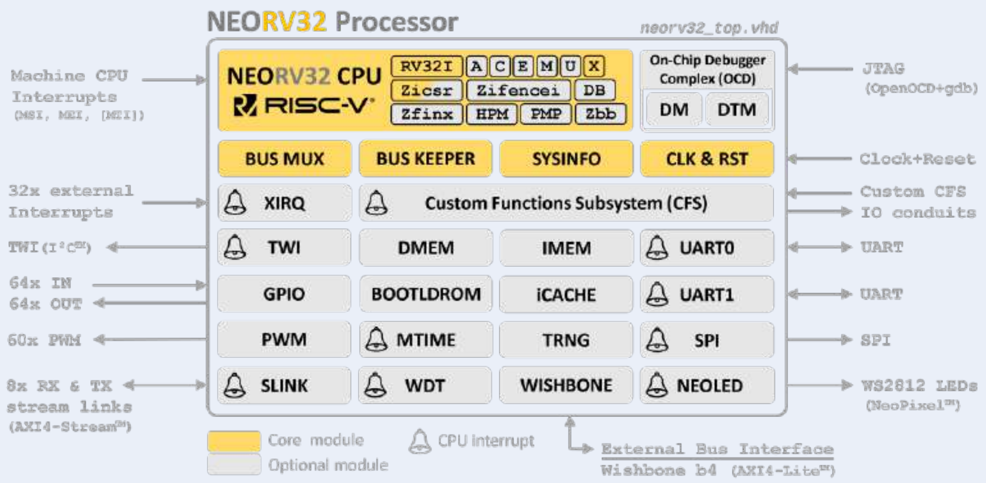


Figure 3: Block diagram showing the NEORV32 function blocks (Source: Github / Nolting, S. [20]).

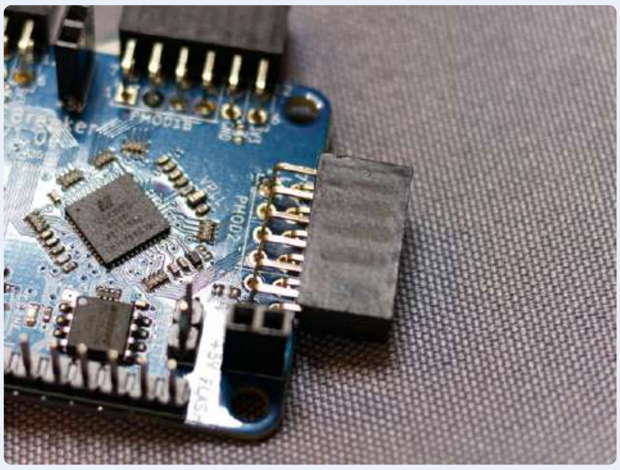


Figure 4: The iCE40UP5K in QFN outline measuring 7x7 mm.

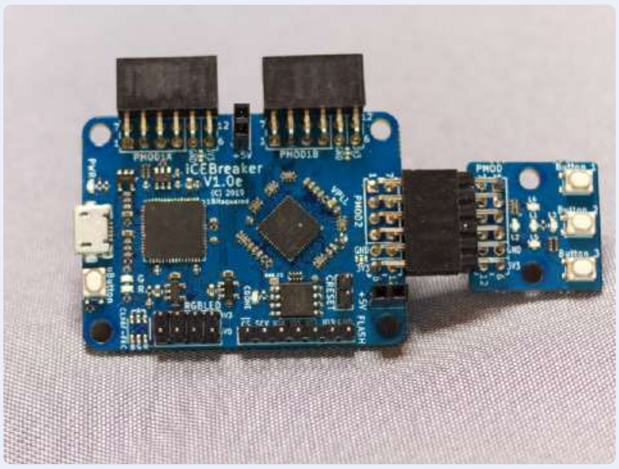


Figure 5: iCEBreaker-Board with PMOD-Header.

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 I^{2}C 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, I^{2}C 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-xxxxxxxx.tgz*. In a terminal, this file is now unpacked using `tar -xvzf oss-cad-suite-linux-x64-xxxxxxxx.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`.

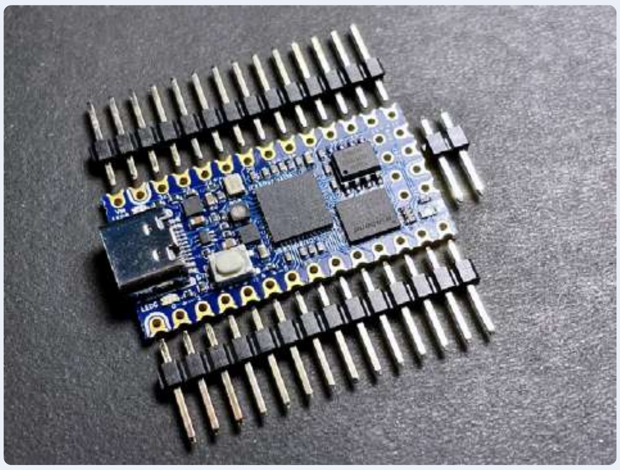


Figure 6: iCEBreaker Bitsy (Source: https://cdn.shopify.com/s/files/1/1069/4424/products/IMG_3859_large.jpg / 1BitSquare).

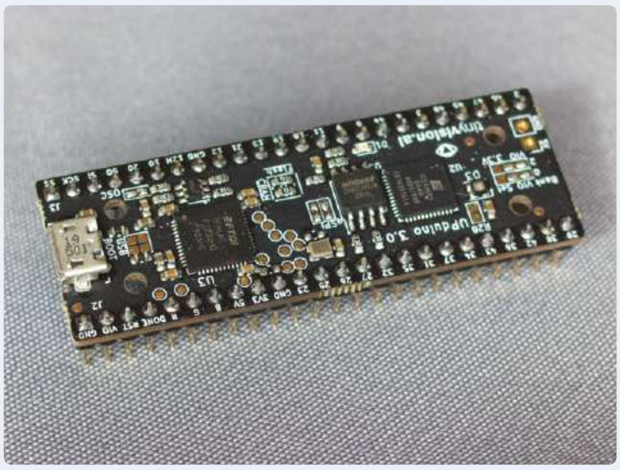


Figure 7: The UPduino V3.0 with pin header strips fitted.

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 zlib1g-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 compiliation.

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/enviroment* file. Using `sudo nano /etc/enviroment` 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:

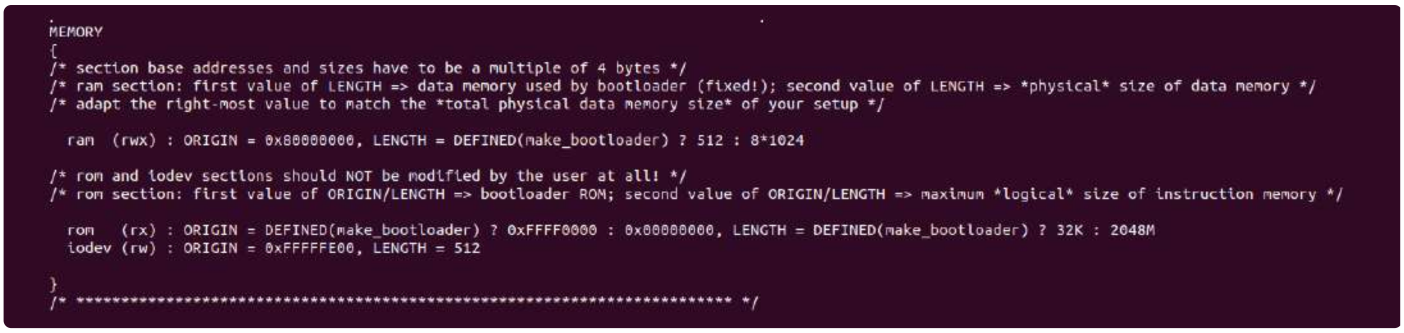
```
ACTION=="add", ATTR{idVendor}=="0403",
   ATTR{idProduct}=="6010", MODE:="666"
ACTION=="add", ATTR{idVendor}=="0403",
   ATTR{idProduct}=="6014", MODE:="666"
```

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

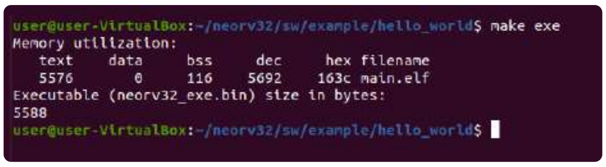
In its de-energized state, the FPGA is not configured. On power up, the iCE40UP5K reads the connection description from an external SPI flash. This description of the internal connections must first be stored into the SPI flash on the UPDuino V3.0 or the iCEBreaker.

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.

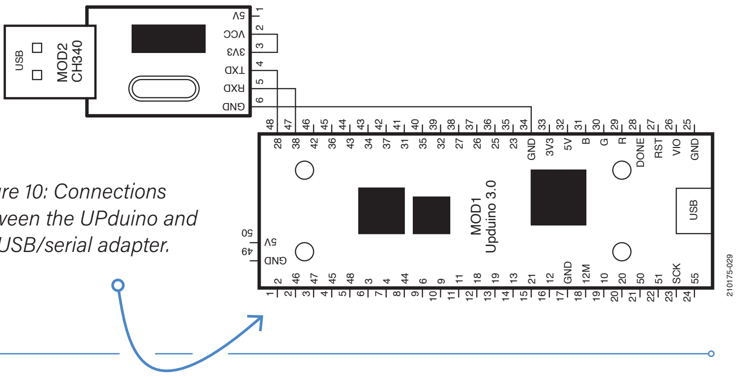


Figure 10: Connections between the UPduino and the USB/serial adapter.

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 `iceprog ~/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, I^{2}C, 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`.

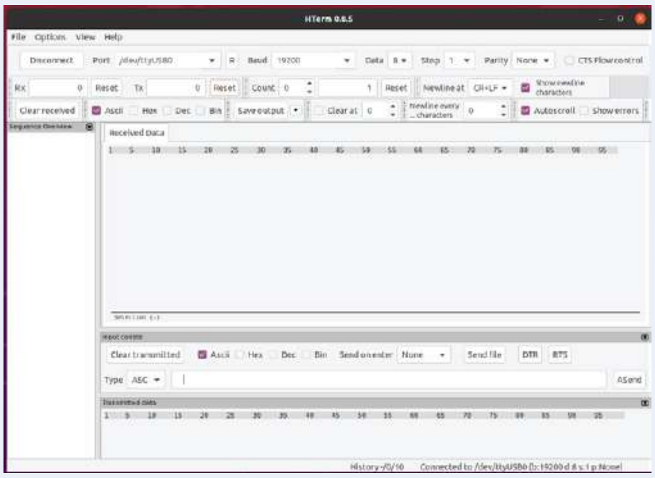
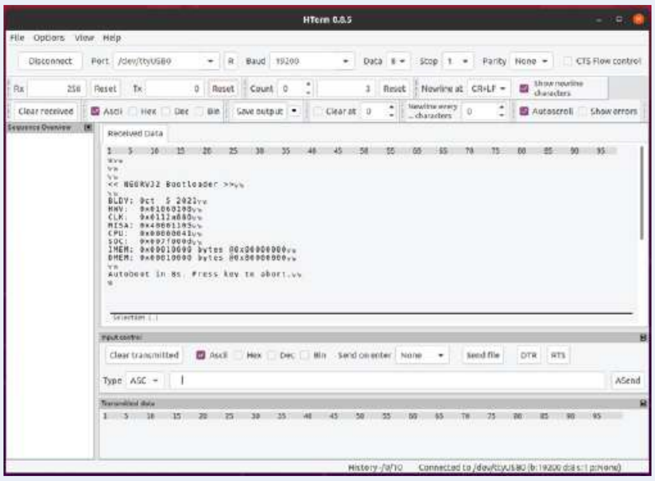


Figure 11: An open HTerm window.



```
\r\n
\r\n
\r\n
<< NEORV32 Bootloader >>\r\n
\r\n
BLDV: Oct  5 2021\r\n
HWV:  0x01060108\r\n
CLK:  0x0112a880\r\n
MISA: 0x40801105\r\n
CPU:  0x00000041\r\n
SOC:  0x007f000d\r\n
IMEM: 0x00010000 bytes @0x00000000\r\n
DMEM: 0x00010000 bytes @0x80000000\r\n
\r\n
Autoboot in 8s. Press key to abort.\r\n
\n
```

Figure 12: The NEORV32 bootloader.

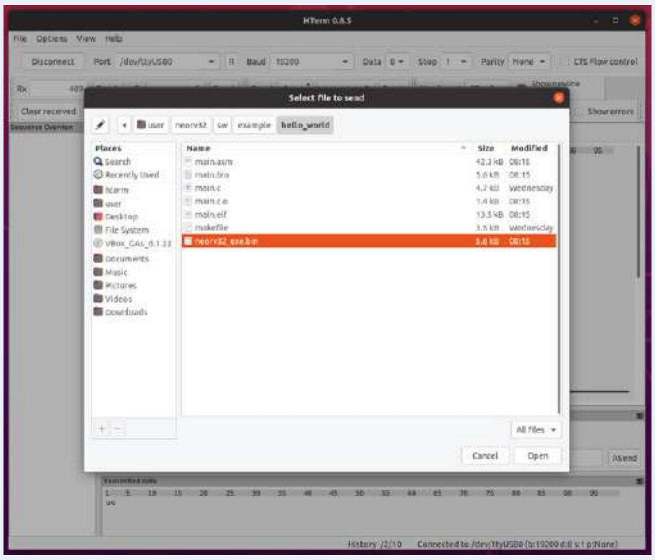


Figure 13: Dialog to upload *neorv32_exe.bin*.

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

Figure 14: A successful upload.

```
<< NEORV32 Bootloader >>\r\n
\r\n
BLDV: Oct  5 2021\r\n
HWV:  0x01060108\r\n
CLK:  0x0112a880\r\n
MISA: 0x40801105\r\n
CPU:  0x00000041\r\n
SOC:  0x007f000d\r\n
IMEM: 0x00010000 bytes @0x00000000\r\n
DMEM: 0x00010000 bytes @0x80000000\r\n
\r\n
Autoboot in 8s. Press key to abort.\r\n
Aborted.\r\n
\r\n
Available CMDs:\r\n
 h: Help\r\n
 r: Restart\r\n
 u: Upload\r\n
 s: Store to flash\r\n
 l: Load from flash\r\n
 e: Execute\r\n
CMD:> u\r\n
Awaiting neorv32 exe.bin... OK\r\n
CMD:> e\r\n
Booting...\r\n
\r\n
\r\n
                                                                                       ##                                       \r\n
                                                                                       ##         ##   ##   ##                  \r\n
 ##     ##   #########   ########    ########   ##      ##   ########    ########      ##       ################                \r\n
####    ##  ##          ##      ##  ##      ##  ##      ##  ##      ##  ##      ##     ##     ####            ####              \r\n
## ##   ##  ##          ##      ##  ##      ##  ##      ##          ##         ##      ##       ##   ######   ##                \r\n
##  ##  ##  #########   ##      ##  #########   ##      ##      #####        ##        ##     ####   ######   ####              \r\n
##   ## ##  ##          ##      ##  ##    ##     ##    ##           ##     ##          ##       ##   ######   ##                \r\n
##    ####  ##          ##      ##  ##     ##     ##  ##    ##      ##   ##            ##     ####            ####              \r\n
##     ##    #########   ########   ##      ##      ##       ########   ##########     ##       ################                \r\n
                                                                                       ##         ##   ##   ##                  \r\n
                                                                                       ##                                       \r\n

Hello world! :)\r\n
```

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_
   spram.vhd devices/ice40/neorv32_dmem.ice40up_spram.
   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.

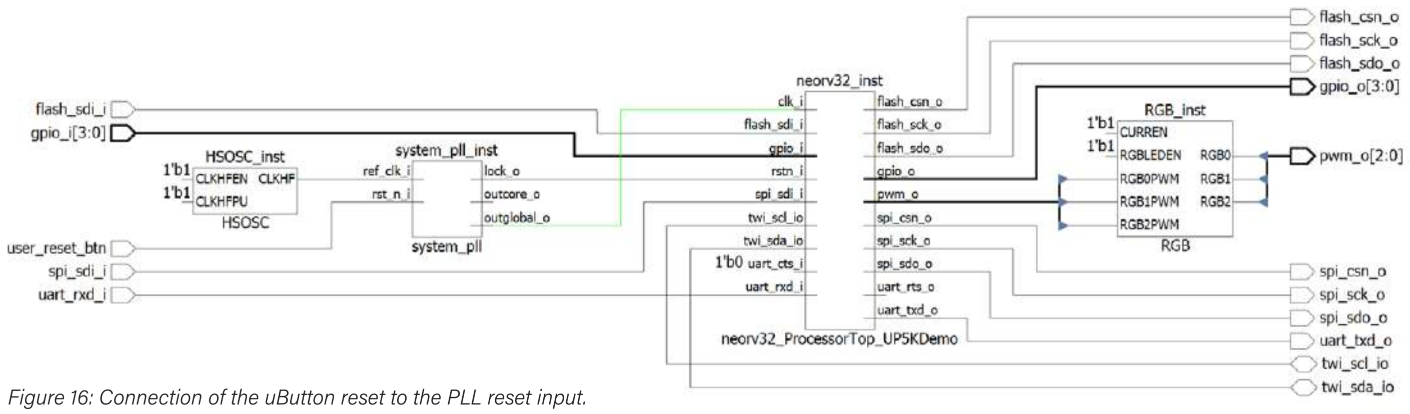


Figure 16: Connection of the uButton reset to the PLL reset input.

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].

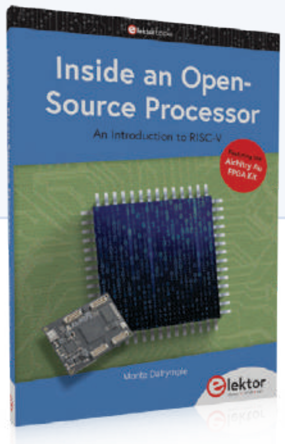
210175-01

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.



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-FEoQ
[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

Under Your Radar

Microcontrollers You Should Know About

By Clemens Valens (Elektor)

The global micrcontroller 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.

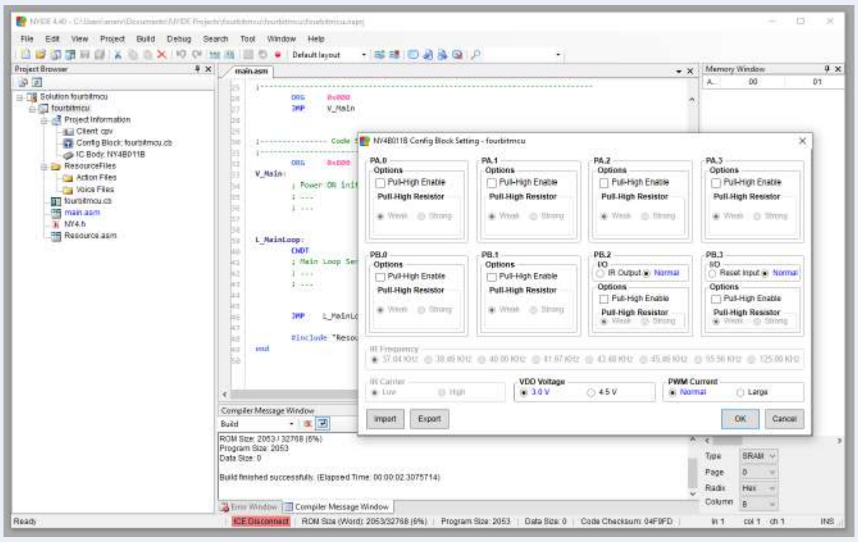
Furthermore, 8051 users have developed a lot of software and know-how during this period, and you don't throw that away just because a better microcontroller comes along.

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 and documented in Chinese only, but they



```
ORG     0x0000
JMP     V_Main

V_Main:
        ; Power ON Init
        ; ...
        ; ...

L_MainLoop:
        CWDT
        ; Main Loop Ser
        ; ...
        ; ...

        JMP     L_MainLo

        #include "Resou
end
```

```
ROM Size: 2053 / 32768 (6%)
Program Size: 2053
Data Size: 0

Build finished successfully. (Elapsed Time: 00:00:02.3075714)
```

```
Ready    ICE Disconnect | ROM Size (Word): 2053/32768 (6%) | Program Size: 2053 | Data Size: 0 | Code Checksum: 04F9FD |    ln 1    col 1    ch 1    INS
```

Figure 1: The NYIDE 4.40 from Nyquest shows that 4-bit microcontrollers can have modern IDEs too.

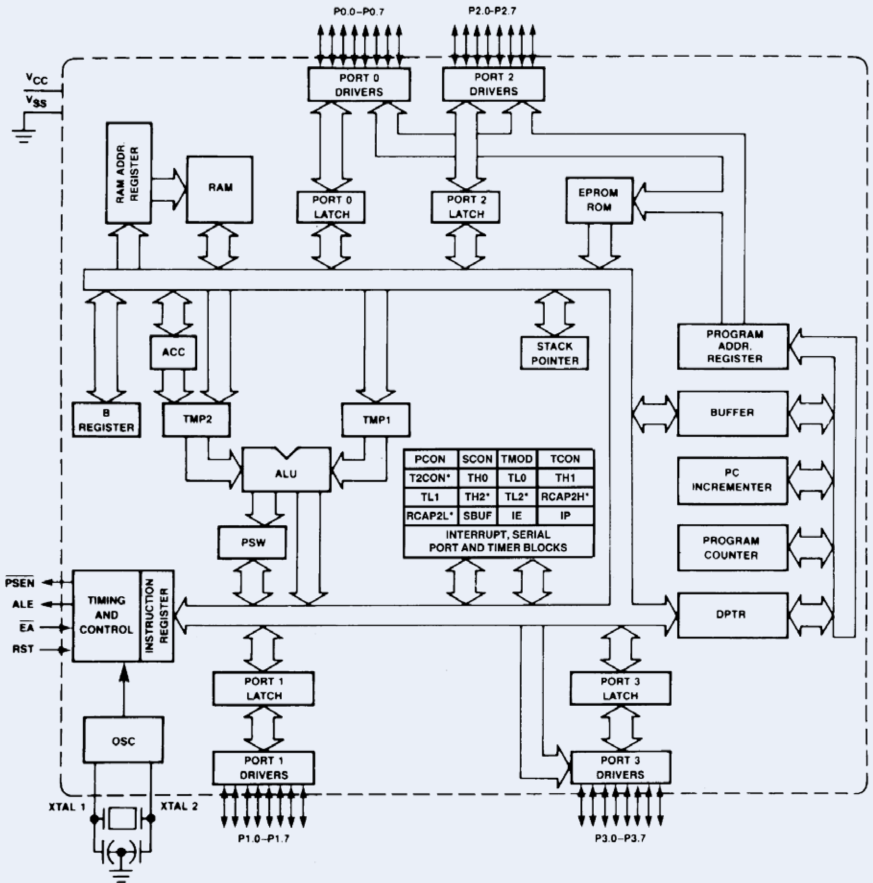


Figure 2: This is the basic architecture of what may very well be the world's most used microcontroller core, the 8051. (Source: Intel)

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].

Figure 3: The Commodore 64, one of the famous home computers from the 1980s, has a 6502 inside. (Source: ralfsfotoseite @ Pixabay)

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 Padauk, MDT, and Holtek.

Padauk

Padauk makes 3-cent one-time programmable (OTP) and flash-based MCUs. The particularity of their devices is the archi-

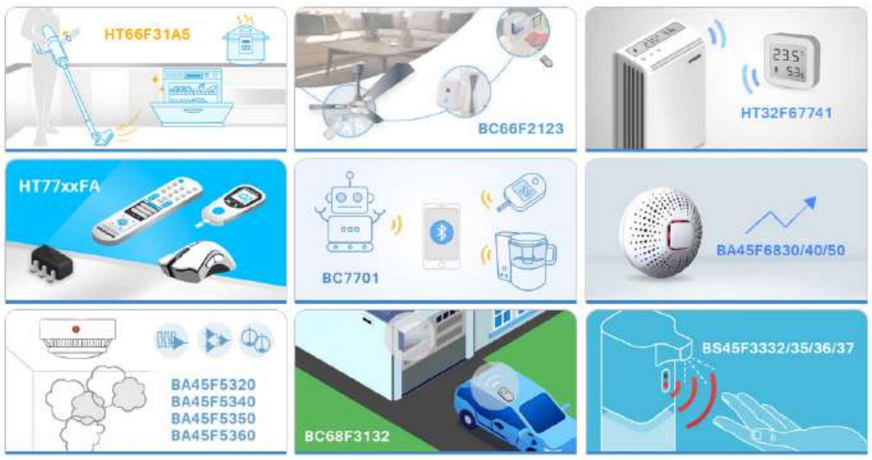


Figure 4: A few examples of applications made possible by ultra-low-cost microcontrollers. (Source: Holtek.com)

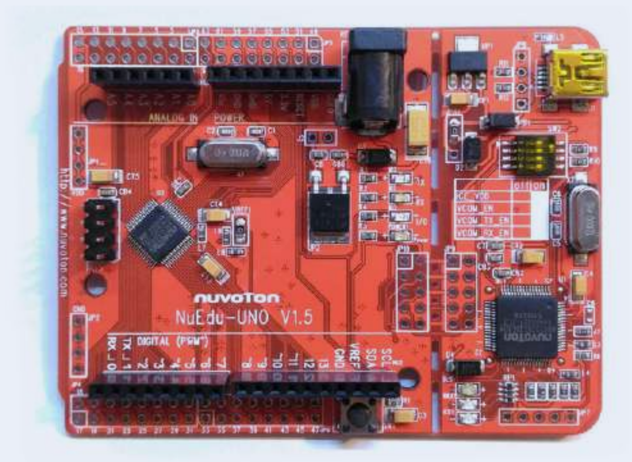


Figure 5: The Nuvoton NuMaker Uno board has a detachable NuLink programmer/debug module. (Source: https://danchouzhou.blogspot.com)

tecture 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.

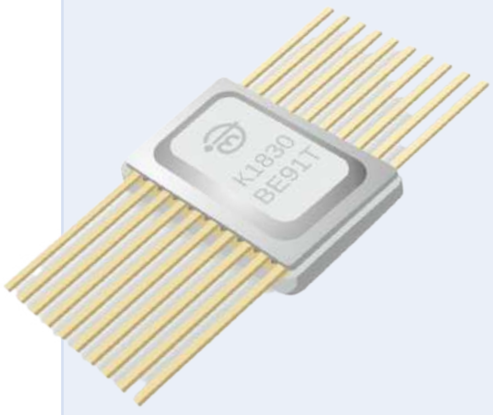


Figure 6: The K1830BE91T from NIIET has an 8051 core and is functionally equivalent to Microchip's AT89C2051. (Source: https://niiet.ru)

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.

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.

NIIET 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 NIIET 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.

QUIZ

Both the 4004 and 4040 were designed by Federico Faggin. Which other famous processors did he design as well?

Intel 8080, Zilog Z8 & Z80

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

What Is **E-Paper Display?**

How Does It Work?

Contributed by Ynvisible

Electronic paper, or an e-paper display, has many benefits over traditional display technologies like LCD and LED. As a result, e-paper display technology is widely used across different sectors. But, what is e-paper display technology and how does it work? In this article, we will look at how electronic paper works and its advantages.

What Is E-Paper?

E-paper displays can be made using different types of technology, such as electrophoresis, electrowetting, and electrochromism [1]. At Ynvisible, we use electrochromism, as it allows us to use highly scalable, flexible, and cost-effective screen-printing production processes, whereas some display companies, such as E Ink, use electrophoresis (see below).

Electronic paper or e-paper is a display technology that mimics the appearance of ordinary ink on paper to produce a low-power, paper-like display. E-paper displays typically have a so-called image memory, which means they do not need much power to maintain the display; they mostly need power when there is a change in the display. Therefore, e-paper displays are best for low-frequency switching displays, such as signage and labels.

E-paper displays reflect light rather than emit light, making them very comfortable to read even under direct sunlight. They are extremely popular because of their low power consumption, high reflectivity, high contrast, readability, thickness, and wider viewing angle. E-paper displays are widely used in e-readers, real-time bus arrival information, electronic shelf label (ESL) segments, digital menu boards, traffic signs and logistics monitoring devices.

How Does E-Paper Work?

Several e-paper technologies create e-paper displays, and each works differently.

Electrophoretic Displays (e.g., E Ink)
Electrophoretic displays from, for instance, E Ink, contain millions of tiny microcapsules. These microcapsules are filled with a clear fluid having small particles of different colors and electric charges. On applying a positive charge, negatively charged particles move to the top, and positively charged particles move to the bottom of the microcapsule and vice versa. This movement of the particles to the top and bottom makes the surface display a specific color.

Electrowetting Displays (e.g., Etulipa)
Etulipa is one of the companies employing electrowetting display technology, which uses the liquid's surface tension to create electrowetting displays. An electrowetting display contains several tiny cells of transparent polar liquid and a colored oil covering a hydrophobic surface. The oil contracts into a small droplet by applying a low voltage to the cells. It then creates an open or closed optical switch capable of displaying text, art, photos, or even video.

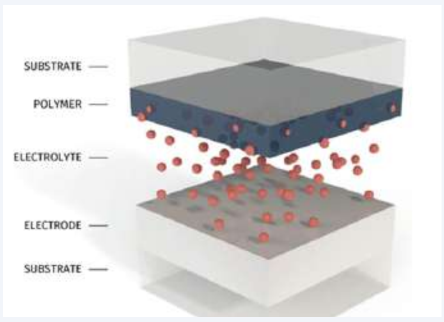


Figure 1: Ynvisible's displays are produced using processes that are highly scalable and cost-effective.

Figure 2: Ynvisible's displays are flexible and lightweight.

Figure 3: The most popular application of e-paper is e-readers.

Figure 4: Electronic paper is perfect for digital signage and public information displays.

Figure 5: E-paper displays are widely used in supply chain management to track and monitor goods in real-time.

Figure 6: E-paper can also be used to counter the piracy of products.

Figure 7: Electronic paper displays provide a cheaper alternative to traditional paper label methods.

Electrochromic Display (e.g., Ynvisible)
Electrochromic displays are produced using electrochromism, a process where material changes color when applying an electric current. The electrochromic material undergoes chemical oxidation and reduction on applying voltage in the presence of an electrolyte. This results in a color change, which can be finely tuned. Ynvisible's electrochromic displays are ultra-energy-efficient, and one coin cell battery can power them for around 50 years.

Ynvisible's displays are produced using screen-printing processes that are highly scalable and cost-effective (see **Figure 1**), making them a more cost-effective solution to other e-paper technologies.

Key Features of E-Paper
The five key features of e-paper technology are:

1. Bistability
E-paper displays have various levels of bistability, which means that the display retains an image without requiring a power source. Ynvisible displays can retain an image for 15 minutes before a small refresh pulse is required. We call this semi-bistability.

2. Low Power Consumption
Since e-paper displays do not need any backlight nor power to retain an image, these displays have extremely low power consumption, making them ideal for off-grid applications, for example.

3. Reflectiveness
E-paper displays reflect light; they do not emit any light, making them incredibly energy-friendly. This also makes these displays ideal for reading in direct sunlight, a common flaw found with other display types.

4. Readability
E-paper displays have excellent readability because of their wider visibility angle and high contrast, making them ideal for important announcements, such as traffic updates.

5. Flexibility
Some e-paper displays are used because of their high flexibility (Ynvisible's displays are flexible enough to curve around a pencil!) and lightweight nature (**Figure 2**). This is a really important factor when shipping is involved, for example.

Ynvisible's e-paper displays [2] are also fully customizable and can be developed to meet your exacting brand requirements.

E-paper displays have been around for many years now. E-paper display mainly started off being used in e-book readers but has found use in many more applications across different industries due to its low-power, low-weight and flexible properties (**Figures 3** to **7**) [3][4].

Get started with Ynvisible's e-paper displays today with our E-Paper Display Kit [5] — it's the best way to trial the technology and work out if it's right for you. Interested developers will find informations about driving the displays at [6].

220273-01

WEB LINKS
[1] What Is Electrochromic Display Technology?: www.ynvisible.com/news-inspiration/what-is-an-electrochromic-display
[2] Cost-Effective Custom Display Prototypes: www.ynvisible.com/news-inspiration/cost-effective-custom-display-prototypes
[3] Anti-Counterfeit & Authenticity Solutions Using Printed Electronics: www.ynvisible.com/news-inspiration/authentication-solutions-printed-electronics
[4] New Cost-Effective Display Applications: www.ynvisible.com/solutions
[5] E-Paper Display Kit: www.ynvisible.com/product/e-paper-display-kit
[6] How to Drive Ynvisible's E-Paper Display: www.ynvisible.com/news-inspiration/how-to-drive-ynvisible-e-paper-display

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

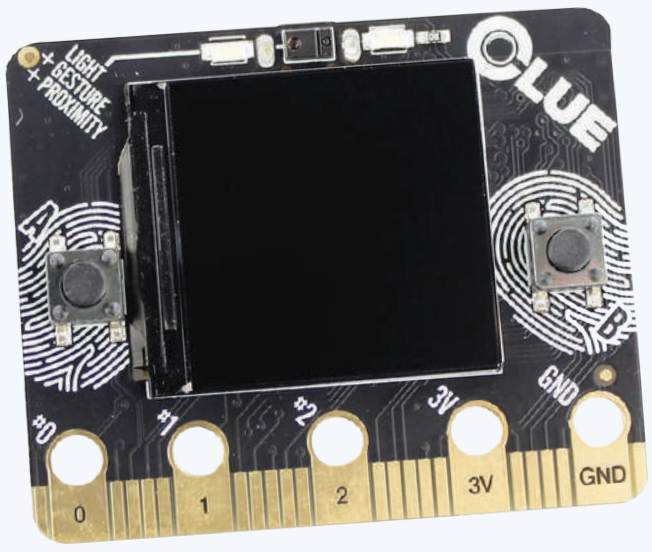


Figure 1: The attacker from the front ...

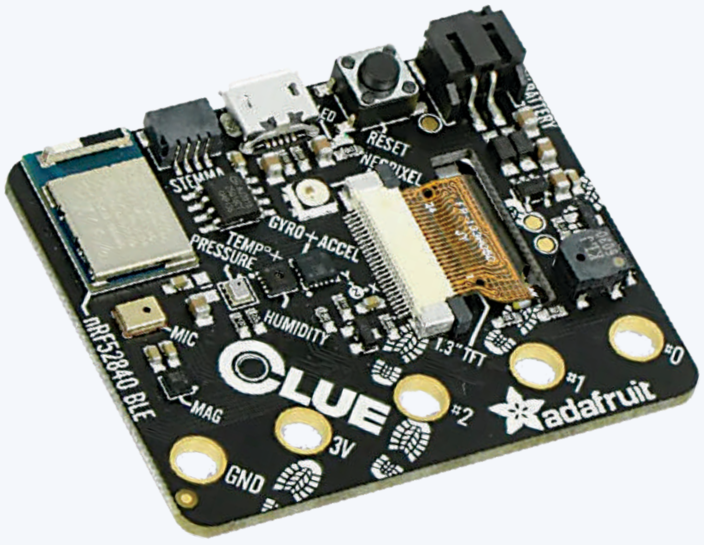


Figure 2: ... and from behind.

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.

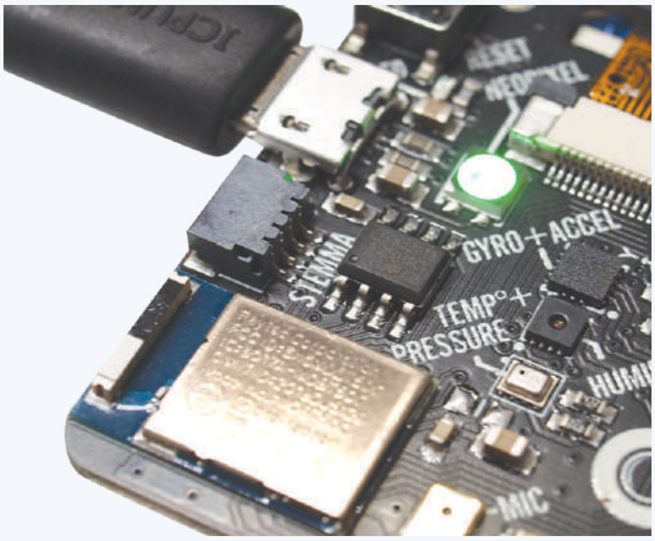
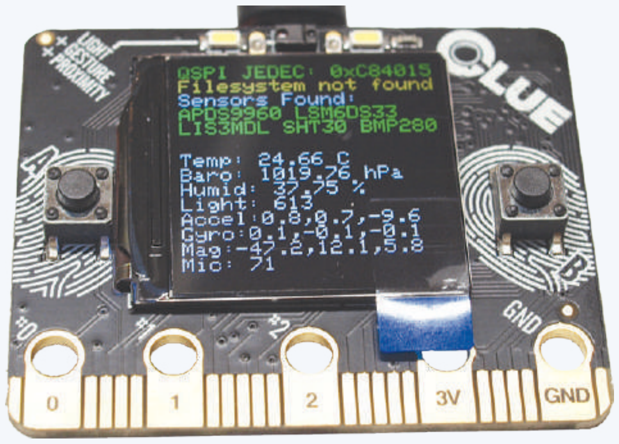


Figure 3: The STEMMA port allows the Adafruit CLUE to connect to extension boards.

A Question of Programming

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



```
QSPI JEDEC: 0xC84015
Filesystem not found
Sensors Found:
APDS9960 LSM6DS33
LIS3MDL SHT30 BMP280

Temp: 24.66 C
Baro: 1019.76 hPa
Humid: 37.75 %
Light: 613
Accel:0.8,0.7,-9.6
Gyro:0.1,-0.1,-0.1
Mag:-47.2,12.1,5.8
Mic: 71
```

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

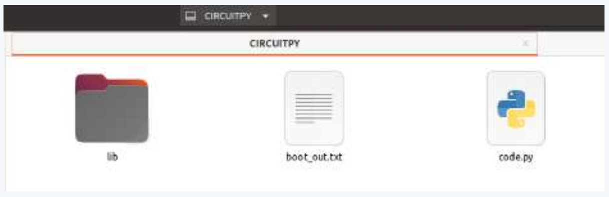


Figure 5: The Python runtime also exposes a virtual drive.

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
```

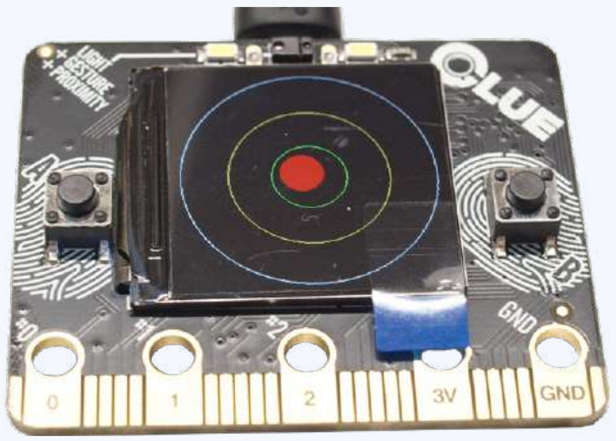
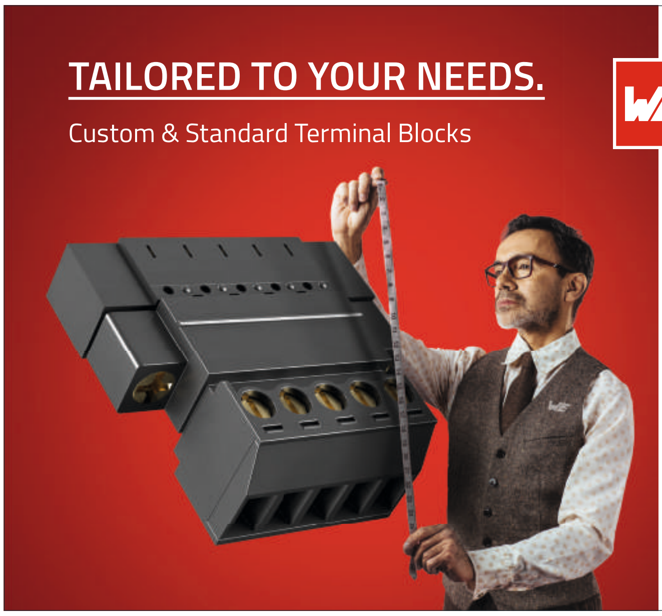


Figure 6: A spirit-level using the on-board sensors.

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



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

embeddedworld
Exhibition&Conference
...it's a smarter world

ynvisible

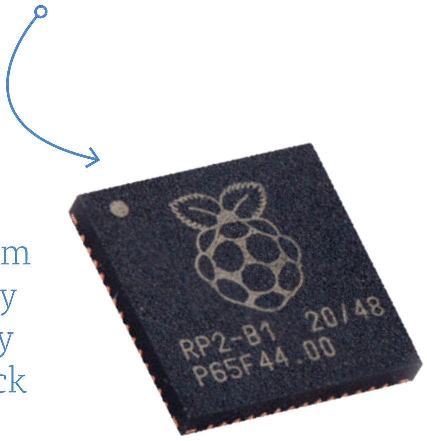
THERE IS A NEW E-PAPER COMPANY IN TOWN

Ynvisible will be showcasing their ultra-low-power and cost-effective e-paper display at **embeded world**

Visit booth 1-349

SCAN TO LEARN MORE!

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.

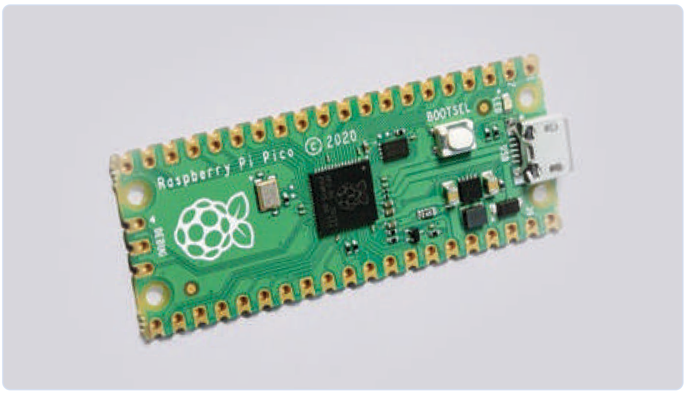


Figure 1: The Raspberry Pi Pico (Source: Raspberry Pi).

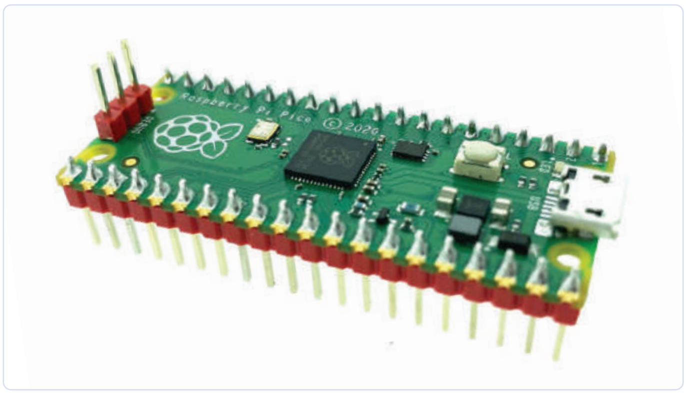


Figure 2: Raspberry Pi Pico with Headers (Source: Elektor).

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.

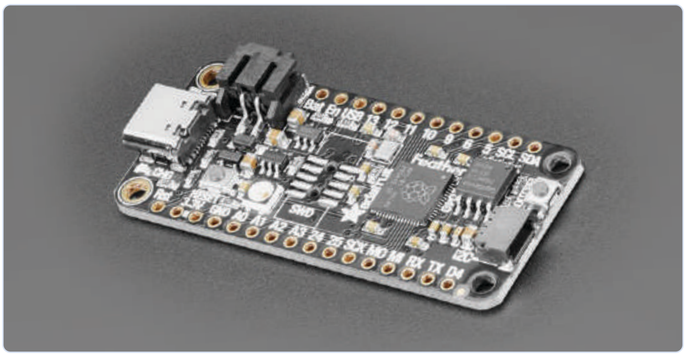


Figure 3: The Adafruit Feather RP2040 (Source: Adafruit).

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 Qwicc 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!

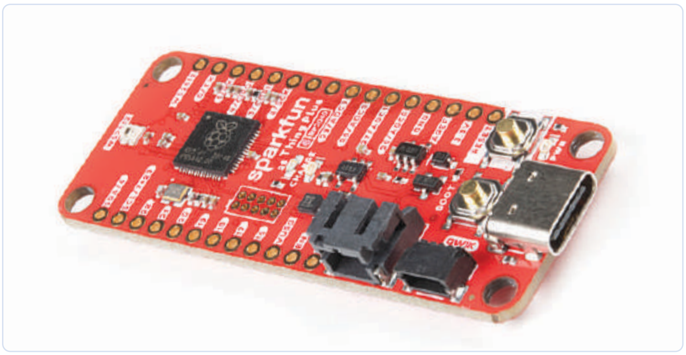


Figure 4: SparkFun Thing Plus RP2040 (Source SparkFun).

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 that 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].

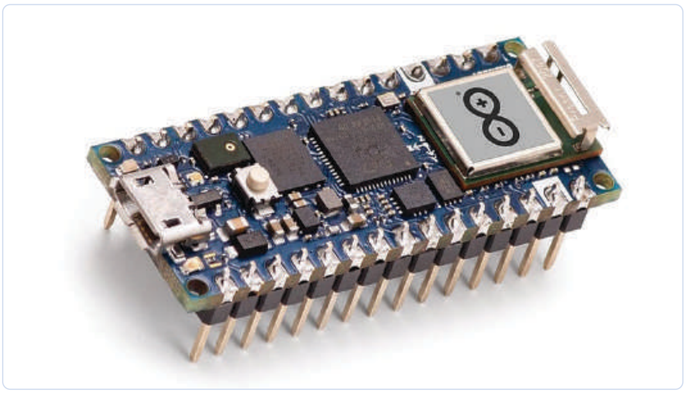


Figure 5: Arduino Nano RP2040 Connect (Source: Arduino.cc).

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.

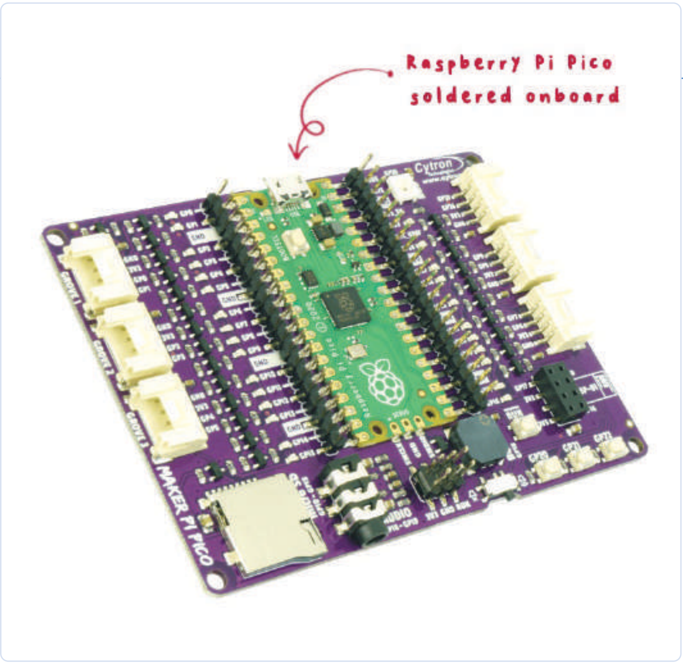


Figure 6: Cytron Maker Pi Pico (Source: cytron.io).

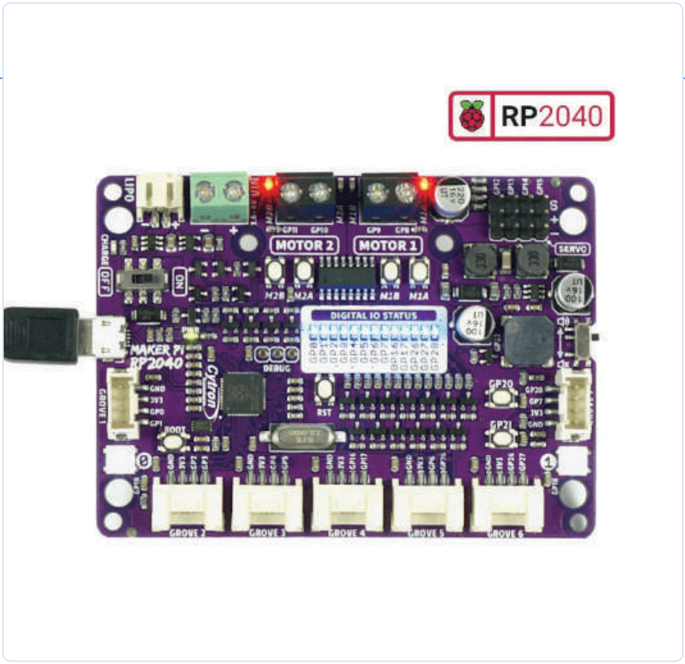


Figure 7: Cytron Maker Pi RP2040 (Source: cytron.io).

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 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.

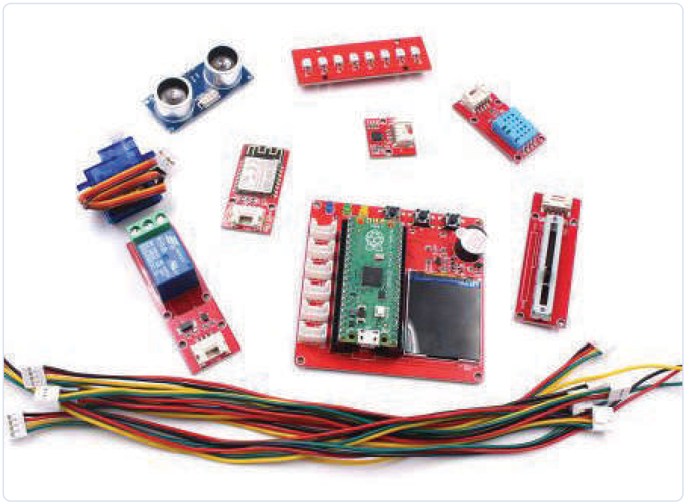


Figure 8: Elektor Raspberry Pi Pico Experimenting Kit (Source: Makerfabs).

Cytron Maker Pi RP2040

Motor control and robotics? With the Raspberry Pi PR2040? 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 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

What's New in Embedded Development?

Rust and Keeping IoT Deployments Updated

Embedded Development

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 a 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

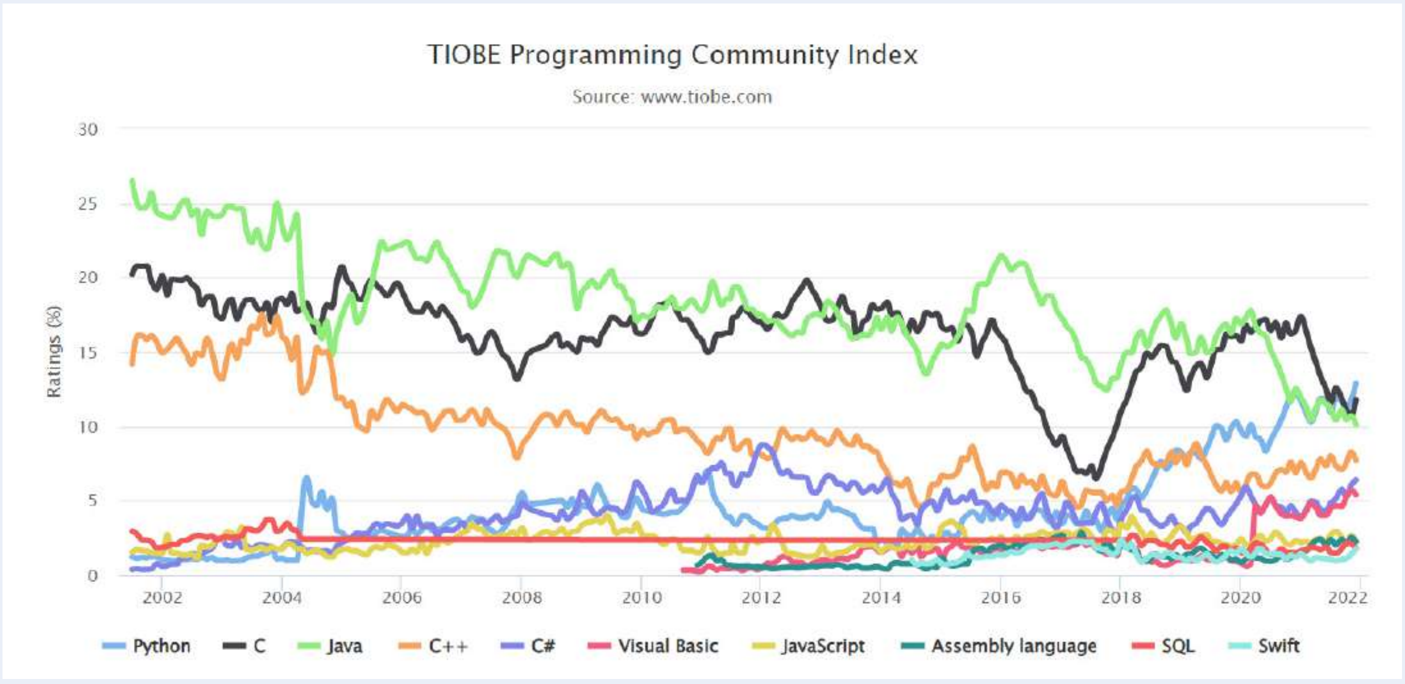


Figure 1: C remains popular as a programming language, regularly placing in the top three in surveys and market analyses. (Source: www.tiobe.com)

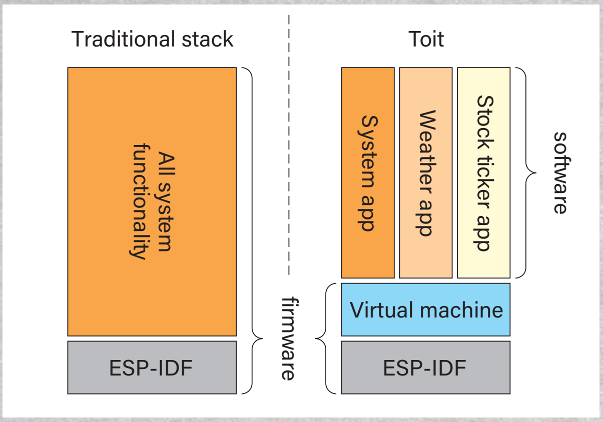
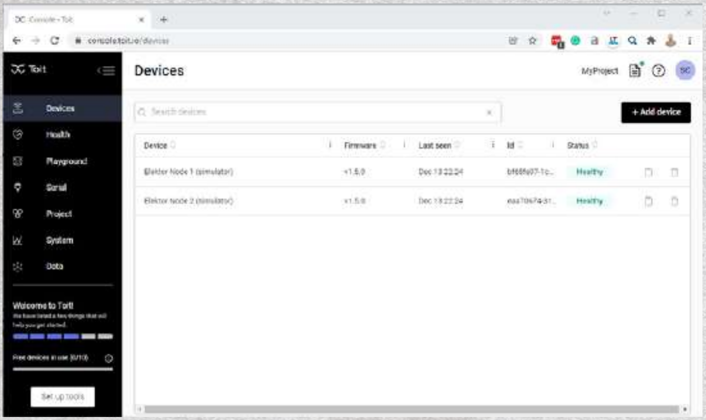


Figure 2: Toit executes IoT code as apps on top of a virtual machine running on an ESP32. (Source: Toit)



Device	Firmware	Last seen	Id	Status
Elektor Node 1 (simulator)	v1.5.0	Dec 13 22:24	bf66fa07-1c...	Healthy
Elektor Node 2 (simulator)	v1.5.0	Dec 13 22:24	ea470674-31...	Healthy

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

suitable crate is *rppal*, 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 rppal dependency to Cargo.toml in a Rust project.

```
[package]
name = "rust_test_project"
version = "0.1.0"
edition = "2021"
[dependencies]
rppal = "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 rppal crate.

```
use std::error::Error;
use std::thread;
use std::time::Duration;
use rppal::gpio::Gpio;
use rppal::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].

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 variablew
        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");
 }
```

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 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

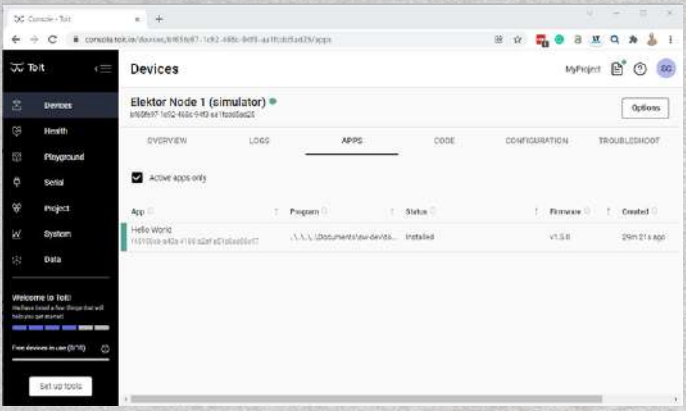


*Figure 4: The **Hello World** app successfully installed on one simulated node.*

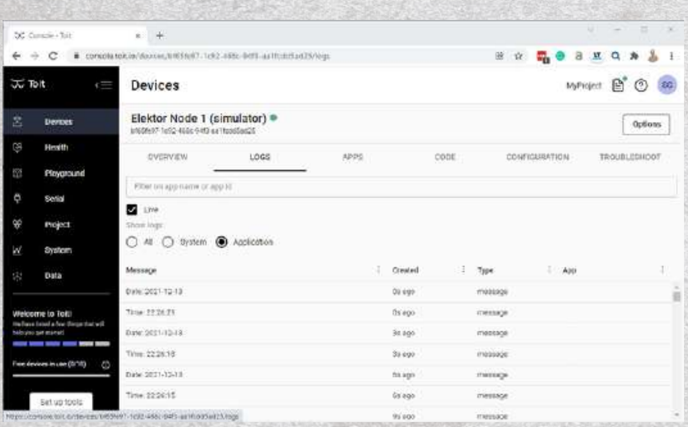


Figure 5: The LOGS output shows the time and date being output from the Toit app every three seconds, as defined in the YAML file.

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.

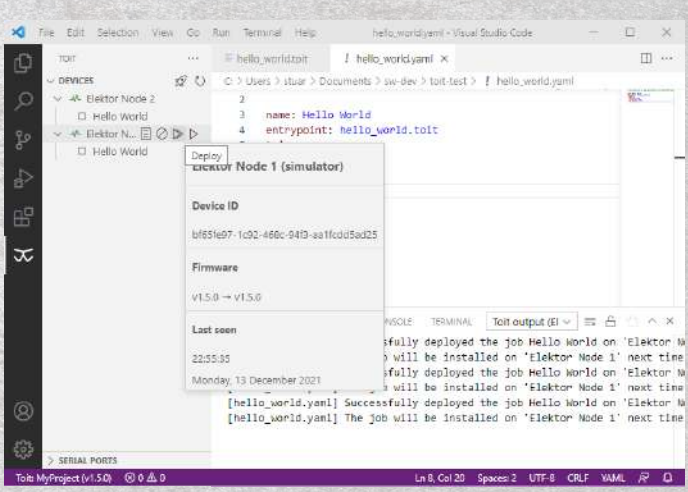
Toit, by contrast, solves a major headache: keeping remote IoT nodes up-to-date, deploying new features to the user base, and all without

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"
```


File Edit Selection View Go Run Terminal Help
hello_world.yaml - Visual Studio Code
TOIT
DEVICES
Elektor Node 2
Hello World
Elektor N...
Hello World
Deploy
hello_world.toit
hello_world.yaml
C: > Users > stuar > Documents > sw-dev > toit-test > hello_world.yaml
CONSOLE TERMINAL Toit output (El
SERIAL PORTS
Toit: MyProject (v1.5.0) ⊗ 0 ⚠ 0
Ln 8, Col 20 Spaces: 2 UTF-8 CRLF YAML

```
name: Hello World
entrypoint: hello_world.toit
```

Elektor Node 1 (simulator)
Device ID
bf651e97-1c92-460c-94f3-aa1fcdd5ad25
Firmware
v1.5.0 → v1.5.0
Last seen
22:55:35
Monday, 13 December 2021

```
sfully deployed the job Hello World on 'Elektor N
) will be installed on 'Elektor Node 1' next time
sfully deployed the job Hello World on 'Elektor N
) will be installed on 'Elektor Node 1' next time
[hello_world.yaml] Successfully deployed the job Hello World on 'Elektor N
[hello_world.yaml] The job will be installed on 'Elektor Node 1' next time
```

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

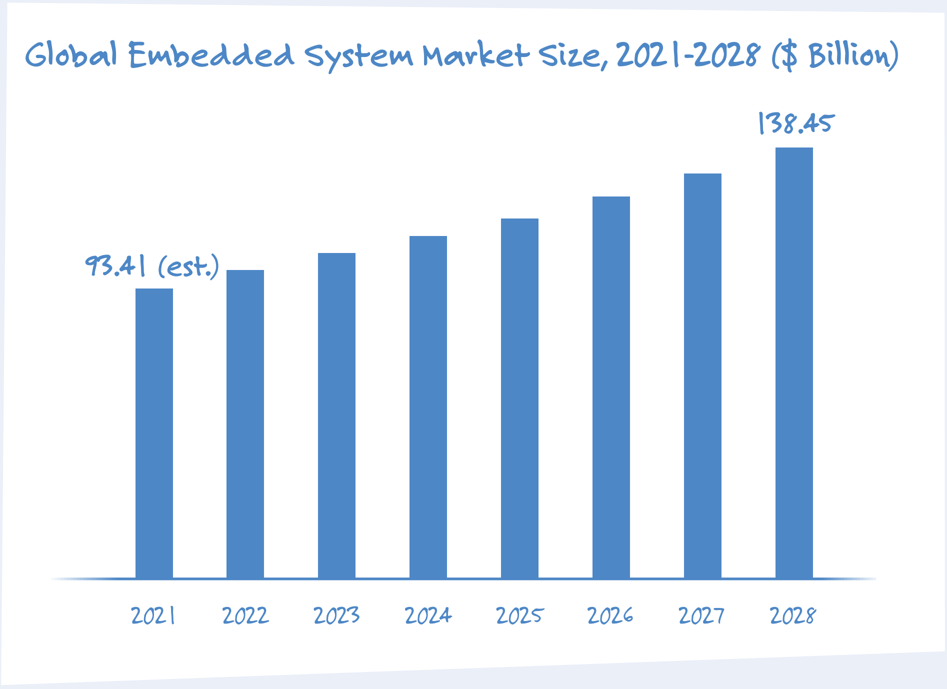
Elektor infographic

By Robert van der Zwan

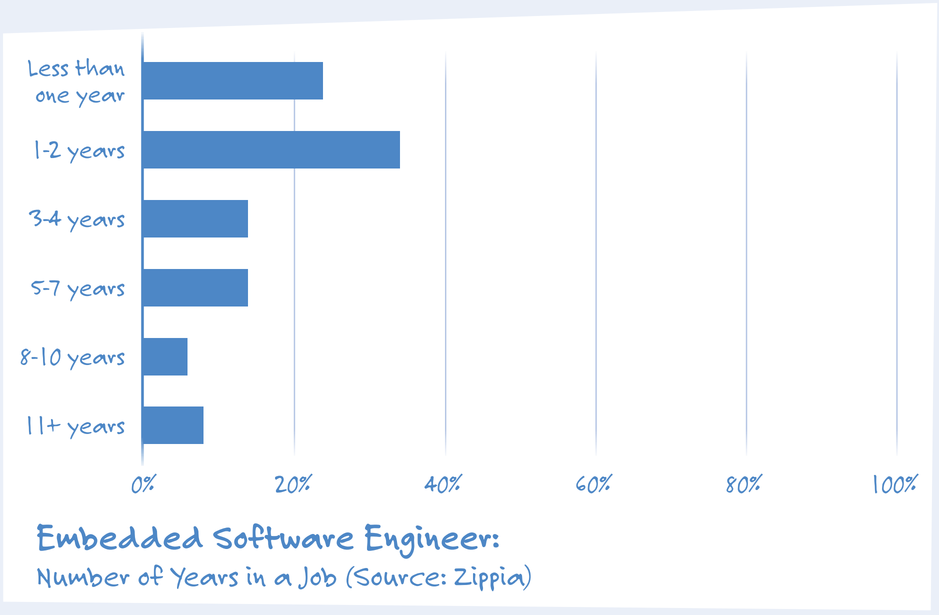
Embedded: Its Growth Is Solidly Embedded Indeed

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)



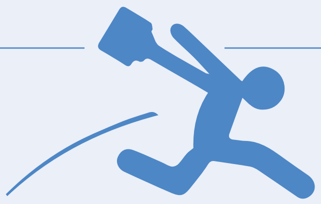
Embedded Software Engineer? You Are in the Driving Seat



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.

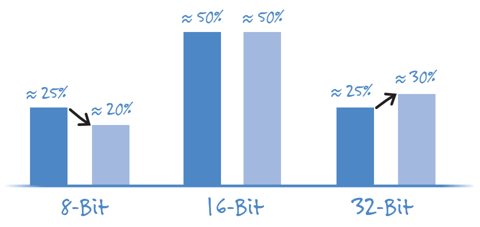
(Sources: AppliedLogix, Zippia)

8-Bit MCUs Falling Somewhat Behind

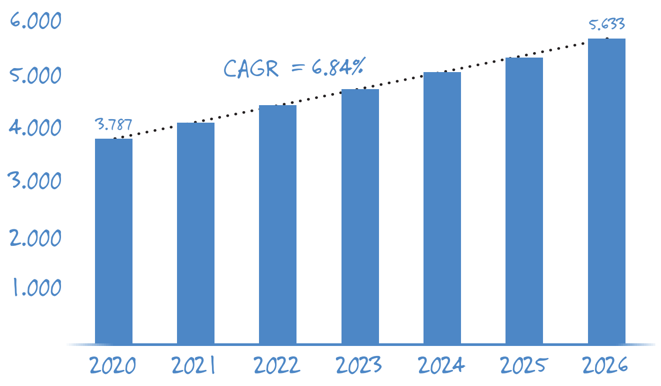
Although 8-bit microcontrollers are here to stay, even at the end of this decade, it is also true that the 32-bit MCUs are making more headway than originally expected. The fact that 32-bit controllers are gaining ground most likely has to do with the fast growth of real-time embedded systems. Whereas 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)

Market Share Microcontrollers, 2021 vs. 2027



Global Embedded Security Market, 2020-2026 ($ Billion)
(Source: Knowledge Sourcing Intelligence)



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 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 that 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 telematics, 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 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

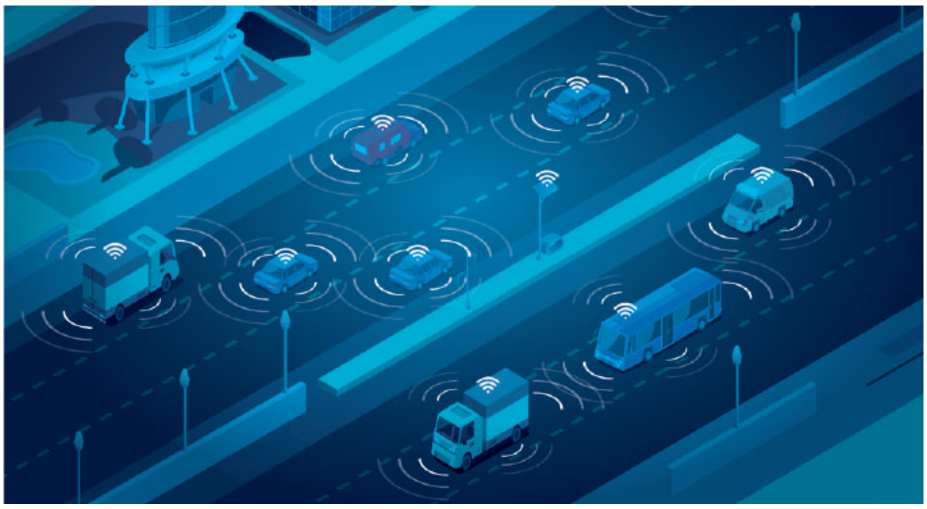
Most of the 5G network roll-out has focused on upgrading 4G/LTE using 3GPP's 5G New Radio Non-Standalone (5G NR NSA), Release 15, specifications, which has enabled the launch of a limited range of 5G services. However, the true potential of 5G is based on the deployment of 3GPP's Release 16 and, further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

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.

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



Your First Steps with an **ESP32-C3** and **the IoT**

A Wi-Fi Button and Relay

By Mathias Claußen (Elektor)

The IoT is not a closed book of hidden secrets. Powerful controllers like the new ESP32-C3 and newbie-friendly development environments like the Arduino IDE make developing small projects a piece of cake.

When we speak of the Internet of Things (IoT), we acknowledge that more and more things in our daily lives are becoming connected to the Internet. It starts with lights, heaters and sensors in the home and continues with cars, traffic lights, shipping containers and much more. Small network-capable components are installed in each of the connected things, which enable exchange of information.

The best way to get to know how to connect your own applications to the IoT is to start out with a simple practical example. In this article we create a link between a Wi-Fi enabled push button and a Wi-Fi enabled relay; the relay can be activated remotely by the button and reports its status back to the push button.

Select the Components

As with all projects, the right components must first be selected. This is where the ESP-C3-12F kit (**Figure 1**), which is available in the Elektor Store, comes into play. This board features a Wi-Fi-enabled ESP32-C3 microcontroller from Espressif. The ESP32-C3 is a replacement for the proven ESP8266. In addition to a modern CPU core, the chip offers a good mix of integrated peripherals that are both beginner-friendly and powerful. (Refer to our review about the ESP32-C3 [1].) An overview of the integrated hardware blocks are depicted in **Figure 2**. In addition to the ESP32-C3, a RGB LED and a USB serial converter are also integrated on the board. We will need two ESP-C3-12F kits for our project.

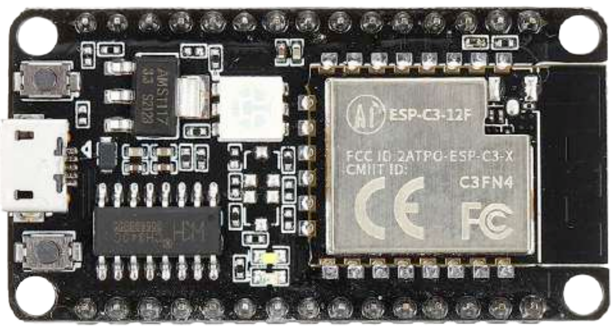


Figure 1: The ESP32-C3-12F Kit.

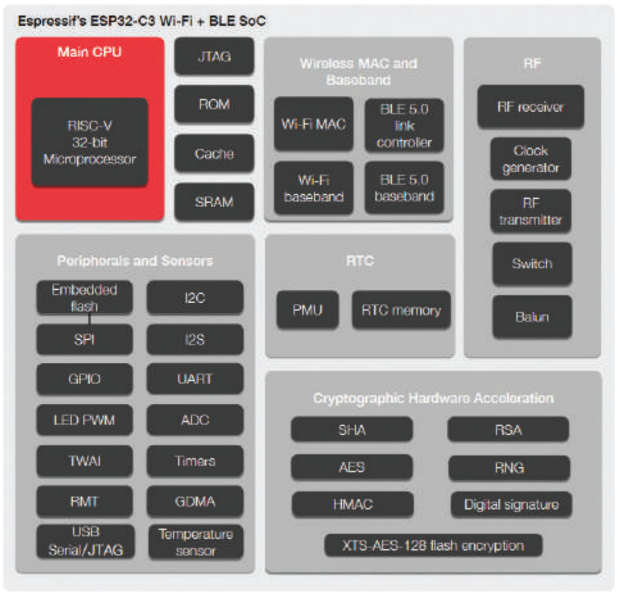


Figure 2: Functional blocks of the ESP32-C3 (Source: ESP32-C3 datasheet).

In addition to the ESP-C3-12F kits, a sensor and an actuator are also required. That's where the Elektor 37-in-1 sensor kit comes in handy, this kit includes 35 sensors (the original version had 37, but two of them contained mercury and have since been omitted on safety grounds). An overview of the sensors of the kit (**Figure 3**) can be found in **Figure 4** and the information document [2]. First of all, we will take the Joystick module and use its push button feature to provide control input to the system. The relay module can now connected to the other microcontroller board and acts as the actuator in the system. A few (female/female) flying leads are required to connect the modules. These are included in the Pimoroni "Mini Breadboards & Jumpers" maker kit. (See the **Related Products** box.)

We also need a computer, such as a Raspberry Pi, which will act as a local server for the IoT devices to exchange their data. An original Raspberry Pi version 1 would in fact do the job, but we recommend at least a Raspberry Pi 2 for this application. On-board Wi-Fi was not included until Raspberry Pi model 3B, so to use earlier versions, a simple Wi-Fi dongle or Ethernet cable will also be required. If you need to buy a small but powerful Raspberry Pi, take a look at the Raspberry Pi Zero 2 W bundle (box). For our purposes, it doesn't however need to be a Raspberry Pi. Any PC running a Linux distribution such as Ubuntu [3] will also prove perfectly adequate.

Before we get into the project, let's take a closer look at how the control and exchange of data takes place in this setup.

MQTT

Any IoT device, whether it's a sensor or actuator, needs to transfer data. For this purpose, we can either go the long way around by developing our own proprietary communication protocol or we can use standard, established protocols. One such system which has become widespread is MQTT. Originally, it was an acronym for "Message Queuing Telemetry Transport," but as the system developed, the title no longer accurately described its function. In 2013, it was officially decided that MQTT would be a label [4].

The MQTT protocol takes care of the exchange of messages using a broker (server) without specifying what the messages look like. You can compare this to sending a letter: the logistics and the format of

Figure 3: The Elektor 37-in-1 sensor kit.

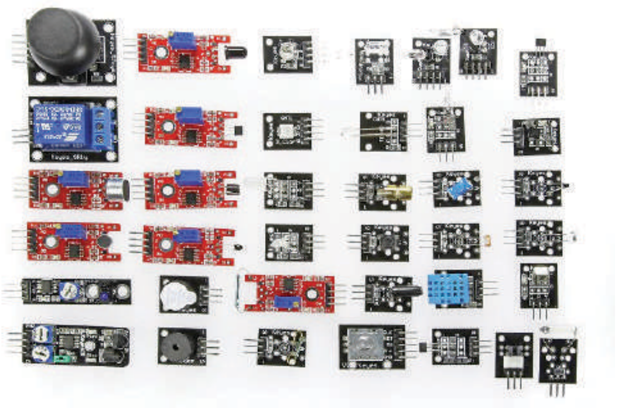
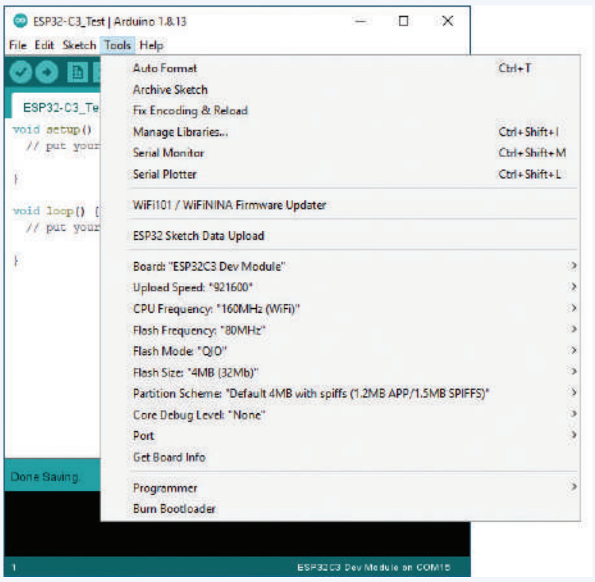


Figure 4: A whole bunch of sensors and actuators are included.


ESP32-C3_Test | Arduino 1.8.13

File Edit Sketch Tools Help

```
ESP32-C3_Te
void setup() {
  // put your

}

void loop() {
  // put your

}
```

Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
ESP32 Sketch Data Upload
Board: "ESP32C3 Dev Module"
Upload Speed: "921600"
CPU Frequency: "160MHz (WiFi)"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Flash Size: "4MB (32Mb)"
Partition Scheme: "Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS)"
Core Debug Level: "None"
Port
Get Board Info
Programmer
Burn Bootloader

Done Saving.

1 ESP32C3 Dev Module on COM15

Figure 5: Configuration of the ESP32-C3 in the Arduino IDE.

the envelope are specified by the postal company, but the message in the envelope and the language in which it is written is entirely up to the user.

Which "language" should we choose to send our messages? Again, there are several options. One popular choice (not only used for MQTT) is Java Script Object Notation (JSON).

JSON

JSON is a lightweight data interchange format for message transfer that is easily generated and interpreted even by small microcontrollers. In addition, JSON text is not only easy to understand for humans, but also easy to write. An overview of the JSON specification can be found on the JSON standard website [5]. JSON is not only used in connection with MQTT, but also in many other areas. JavaScript, the programming language from which JSON is derived, is one of the core technologies on which the World Wide Web is based today. A good introduction to JSON with practical examples can be found on the Mozilla [6] site.

Setting Up the IoT Environment: The MQTT Broker

As with all projects, proper preparation helps avoid unexpected surprises later. To handle MQTT messages, we require a broker device that can either exist on the Internet or we can install one locally on our network. A local broker means we won't need to rely on cloud services for IoT functionality that for our purposes will only be used to transfer messages between locally connected devices. The broker can be built using a "retired" PC or Raspberry Pi, for example. Using Node-RED, we get a complete toolbox for developing network applications that is not just limited to processing MQTT messages. Node-RED [7] has already been used frequently at Elektor to process MQTT data and can be installed quickly on a Raspberry Pi [8] or a PC [9] thanks to the detailed instructions.

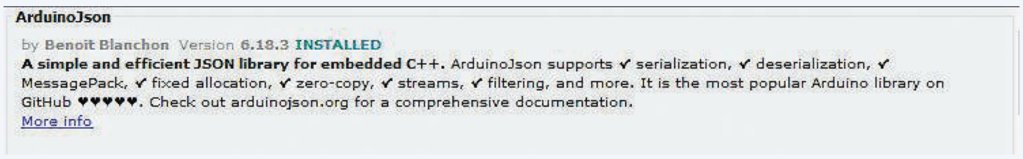
The Arduino IDE

The Arduino IDE is used here as the development environment. The IDE's editor is not the best in its class, but currently offers the most stable support for the ESP32-C3. The Arduino IDE [10] can be downloaded and installed free of charge from the Arduino homepage. The Arduino ESP32 support (as described in the Espressif documentation [11]) must now be installed. The settings for the board are made as shown in **Figure 5**.

In addition to the Arduino ESP32 support, a few libraries will also be required for our first steps. In our example for the ESP32-C3 to be able to send data via MQTT/JSON, we will need to install Nick O'Leary's *PubSubClient* and Benoit Blanchon's *ArduinoJson* libraries. These can be installed using the Arduino IDE Library Manager (**Figures 6** and **7**).

Assembling the Hardware

The two ESP32-C3 based modules are assembled according to the circuit diagrams in **Figure 8** and **Figure 9**. The joystick and relay modules require just three wires connected to the respective ESP32-C3


ArduinoJson
by **Benoit Blanchon** Version **6.18.3** **INSTALLED**
A simple and efficient JSON library for embedded C++. ArduinoJson supports ✔ serialization, ✔ deserialization, ✔ MessagePack, ✔ fixed allocation, ✔ zero-copy, ✔ streams, ✔ filtering, and more. It is the most popular Arduino library on GitHub ♥♥♥♥♥. Check out arduinojson.org for a comprehensive documentation.
More info

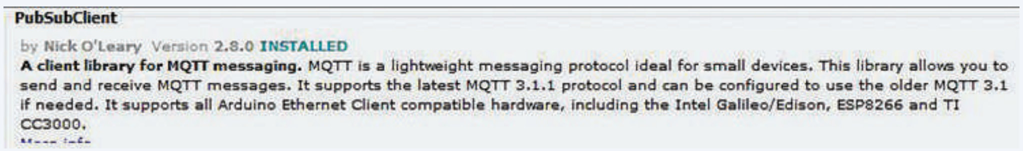

PubSubClient
by **Nick O'Leary** Version **2.8.0** **INSTALLED**
A client library for MQTT messaging. MQTT is a lightweight messaging protocol ideal for small devices. This library allows you to send and receive MQTT messages. It supports the latest MQTT 3.1.1 protocol and can be configured to use the older MQTT 3.1 if needed. It supports all Arduino Ethernet Client compatible hardware, including the Intel Galileo/Edison, ESP8266 and TI CC3000.

Figure 6: PubSubClient in the Arduino Library Manager.

Figure 7: ArduinoJson in the Arduino Library Manager.

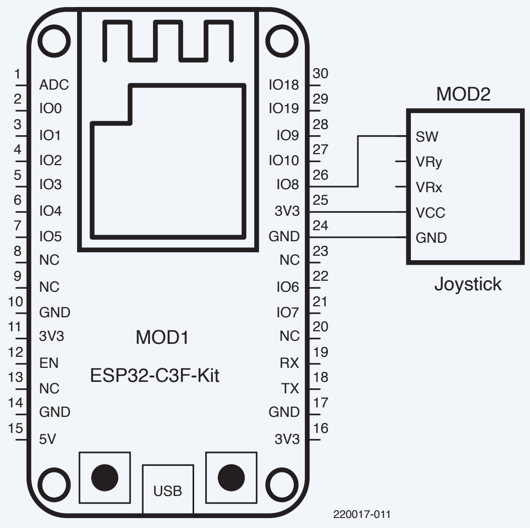


Figure 8: Circuit diagram of the ESP32-C3 and Joystick.

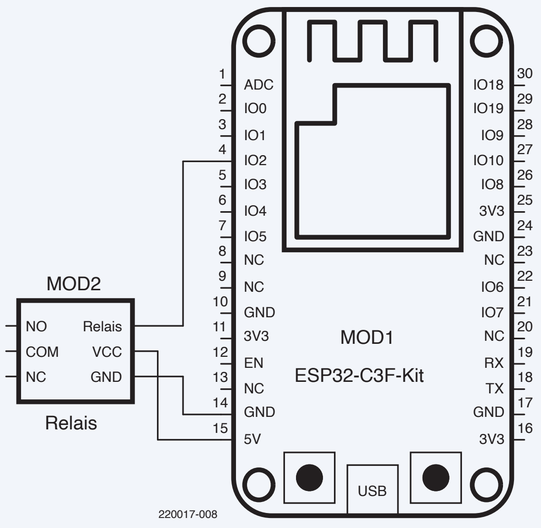


Figure 9: Circuit diagram of the ESP32-C3 and Relay.

board. Note the relay board is powered from the 5 V supply pin. **Figure 10** shows all the modules and controllers connected together.

Software Setup

The source code for this project is also available on GitHub [12]. The sketches for the two ESP32-C3 controllers can be downloaded from there. It will be necessary to enter some information about your local network to these files before we can upload them to the ESP32s. Everything must be set appropriately so that the two controllers can exchange data with the local MQTT broker. For this purpose, `#define` directives are present at the beginning of the two Arduino sketches:

```
#define WIFI_SSID "changeme"
#define WIFI_PASS "changeme"
#define MQTT_SERVER "test.mosquitto.org"
```

These three `#defines` at the beginning of the Arduino sketch must be adjusted for your own network. The SSID and PASSWORD of your network need to be entered in the appropriate position between the quotation marks. The IP address of the Node-RED computer in its own network is specified for the MQTT server. Once both sketches (for the relay and for the button) have been edited they can be uploaded to the respective ESP32-C3. Both ESP32s can then be powered up and the large LED on each board should start flashing white. This indicates that the ESP32-C3 is attempting to connect to the Wi-Fi network. The LED should then light continuously when a board successfully connects. The LED colour will depend on the board's function: the board with the relay connected will light up white. The board with the pushbutton connected will light up red (relay off) or green (relay on) — i.e., according to the relay status.

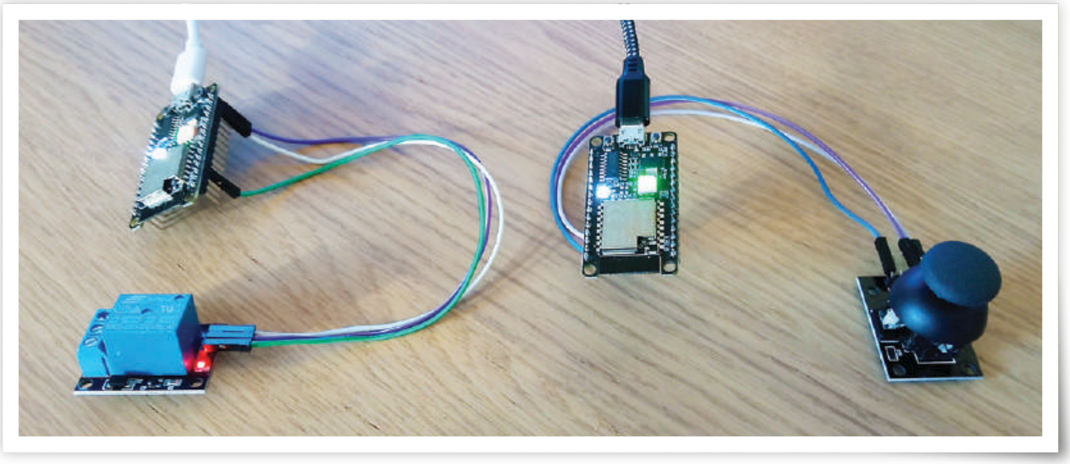


Figure 10: The complete hardware hook up.

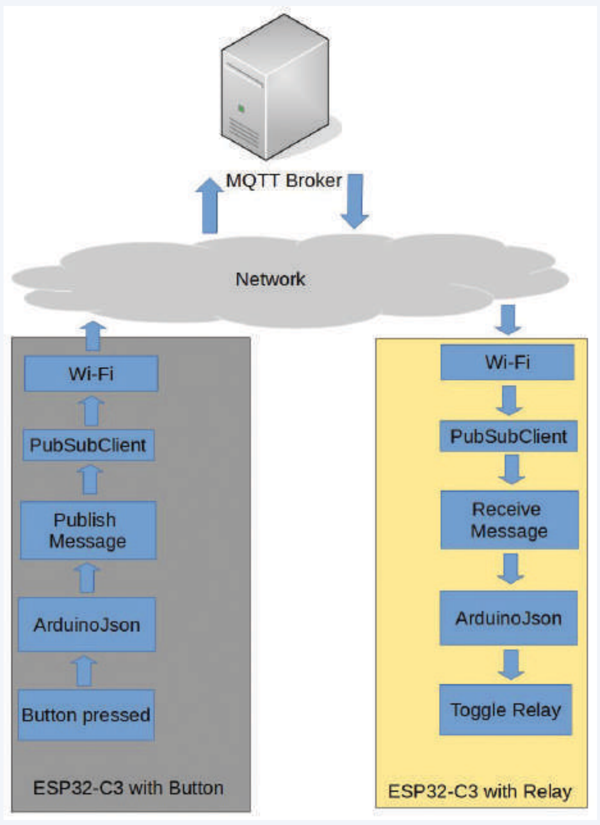


Figure 11: Data transfer to the Broker and back.

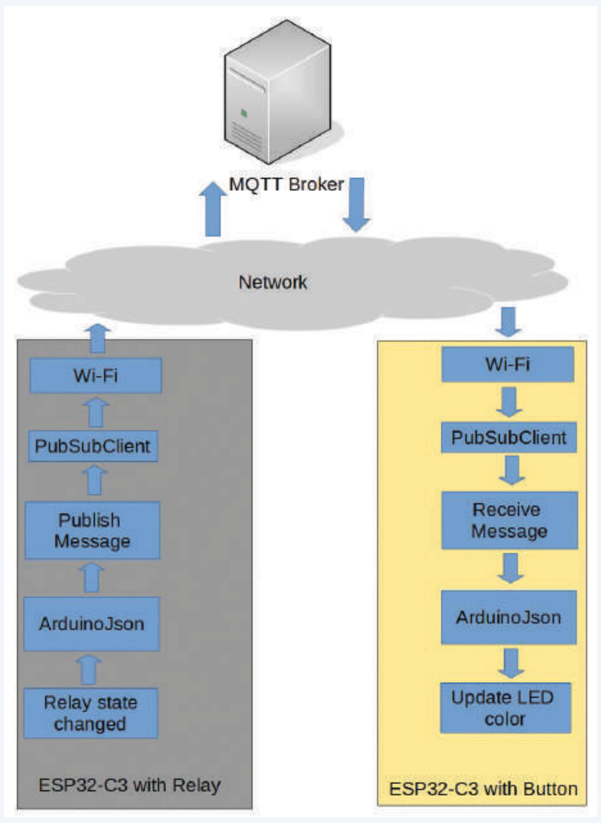


Figure 12: Feedback from the Relay.

This now shows that both ESP32-C3s are operating successfully. Pressing the button will change the state of the relay and the colour of the LED will change from red to green or vice versa. The button of one ESP32-C3 can thus successfully control the relay of the other ESP32-C3, and it also receives feedback on the status of the relay. Time to celebrate. Your first IoT application is running! But how exactly does the data exchange work?

To the Relay and Back Again

First, let's look at the path a button press takes to the relay. **Figure 11** shows how the message is packed layer by layer and then sent to the broker via Wi-Fi. In the source code, this is done using `client.publish(MQTT_TOPIC_OUT, (const uint8_t*)buffer, n, true);`.

Why is this function called `publish` and not simply `send`? This is due to the way the data is later distributed in MQTT. On an MQTT broker, messages are distributed based on a topic; in this case, the topic (`MQTT_TOPIC_OUT`) *"BUTTON"*. When connecting to the MQTT broker, the client (i.e., the ESP32-C3 of our relay) can indicate which topic is of interest (i.e., to "subscribe" to this news channel).

Each participant who has informed the MQTT broker that they are interested in a certain topic will receive the messages sent under this topic. The sender, on the other hand, does not have to worry about distribution. It only sends ("publishes") its messages to the broker.

The ESP32-C3 with the relay subscribes to the *BUTTON* topic in its code using `client.subscribe(MQTT_TOPIC_IN);` , where `MQTT_TOPIC_IN` is "*BUTTON*" here. Every time a message is triggered by the button, it goes to the MQTT broker. It is then delivered to the ESP32-C3 with the relay, causing the relay to toggle.

When the relay changes it state, its controller sends a message to the MQTT broker under the topic *"RELAIS" (RELAY)*, in which the current state (whether on or off) is included. The ESP32-C3 with button, in turn, has subscribed to the *"RELAIS"* topic at the MQTT broker and thereby receives the message with the new status, as shown in **Figure 12**. This causes the LED colour to be set to red or green.

The nice thing about this structure is that a second button and controller can also be integrated into the network to send messages under the topic *"BUTTON"* — just like the first button. The ESP32-C3 with the relay will then respond to the control data from either button and carry out the appropriate switching operation. If you're interested in experimenting further with MQTT, then take a look at Elektor's other projects that use MQTT to send data. Examples are the weather station [13] and the monster LED clock with an external temperature sensor [14]. For more about MQTT and how you can use it to send data to cloud platforms, for example, refer to the series, "My Journey into The Cloud" [15].

An Ideal Platform for the IoT

IoT does not need to be complicated. As we have shown here, the technology is quite mature, and you can develop IoT devices quickly using simple tools and the proper components. Applications can of course be more complex than just a simple push button and relay. From domestic heating control to the front doorbell, there are many practical applications that can benefit from IoT connectivity. The ESP32-C3 is a powerful and inexpensive board that makes an ideal platform to develop your own ideas!

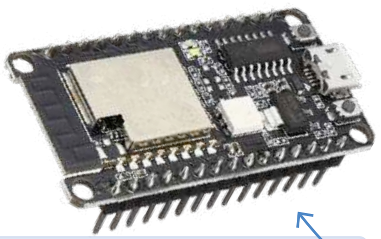
220017-01

Contributors

Idea and Text: **Mathias Claußen**
Editor: **Jens Nickel**
Translator: **Martin Cooke**
Layout: **Giel Dols**

Questions or Comments?

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



WEB LINKS

[1] M. Claußen, "Getting Started with the ESP32-C3 RISC-V MCU," Elektor 1-2/2022: www.elektormagazine.com/210466-01
[2] Elektor 37-in-1 Sensor-Kit Documentation: www.elektor.com/amfile/file/download/file/1170/product/6171/
[3] Ubuntu Linux Distribution: https://ubuntu.com/
[4] OASIS MQTT TC Minutes from 25.04.2013: www.oasis-open.org/committees/download.php/49028/OASIS_MQTT_TC_minutes_25042013.pdf
[5] JSON.org: www.json.org/json-de.html
[6] Mozilla Web Docs: Working with JSON: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON
[7] Node-RED: https://nodered.org/
[8] Node-RED Installation on the Raspberry Pi: https://nodered.org/docs/getting-started/raspberrypi
[9] Node-RED Installation on a PC: https://nodered.org/docs/getting-started/local
[10] Arduino IDE download: www.arduino.cc/en/software
[11] Espressif Arduino-ESP32 Installation instructions: https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html
[12] Elektor GitHub Repository: https://github.com/ElektorLabs/220017-ESP32-C3-and-IoT_First-steps
[13] R. Aarts, "ESP32 Weather Station," Elektor 1-2/2019: www.elektormagazine.com/magazine/elektor-70/42351
[14] M. Claußen, "A Monster LED Clock with Wi-Fi and Temperature Display," Elektor 5-6/2019: www.elektormagazine.com/magazine/elektor-96/42659
[15] J. Nickel, "My Journey into the Cloud," Elektormagazine.com: www.elektormagazine.com/search?query=My+journey+into+the+cloud

A Closer Look at WIZnet WizFi360 Module

Contributed by WIZnet

The WizFi360 is a low-cost Wi-Fi solution which not only is Azure Certified and supports AWS SDK Examples, but also the official Wi-Fi Shield on Arm Open-CMSIS-pack and Keil Studio Cloud; and furthermore, comes in a Raspberry Pi Pico compatible board form-factor. In this article, we will explain the main features of WizFi360 modules and evaluation boards.

What Is the WizFi360?

WizFi360 enables microcontrollers to connect to 2.4-GHz Wi-Fi using IEEE 802.11b/g/n standards. The module has been designed to provide designers with a simple Wi-Fi solution. For easy migration from ESP8266, WizFi360 supports Espressif-like AT command list.

The following attributes of WizFi360 were extracted from its datasheet [1]:

- Wi-Fi 2.4G, 802.11 b/g/n
- Station/SoftAP/SoftAP+Station operation modes
- UART/SPI interface
- “Data pass-through” and “AT command data transfer” mode
- Baudrate up to 2 Mbps with 16 common values
- Support firmware upgrade by UART Download/OTA (via WLAN)
- Industrial grade (operating temperature range: -40°C ~ 85°C)
- CE, FCC, KC, K-MIC(TELEC), RoHS, REACH certification

WizFi360 comes in two options: with onboard pattern antenna or u.fl connector. WIZnet also provides internet-offload (io) modules with various form factors (**Figure 1**). WizFi360io-C is an interface board with SMW200-06 connector and 5 V support, WizFi360io-H is a pin-header type interface board with 2.00 mm pin header.

WIZnet also provides various evaluation boards for easy project prototyping (**Figure 2**). Arduino-compatible WizFi360-EVB-Shield can be used for experiments, tests and verifications of WizFi360. Lately WIZnet released the RP2040-based WizFi360-EVB-Pico — a Raspberry Pi Pico clone but with Wi-Fi connectivity.

Accelerate Internet of Things Solutions

WizFi360 is a small but powerful module that supports SSL using MbedTLS library. With its help, users can easily connect WizFi360 to cloud or MQTTS broker. In October 2019, WIZnet announced it had joined Microsoft Azure Certified for Internet of Things (IoT), ensuring customers get IoT solutions up and running quickly with hardware and software that has been pre-tested and verified to work with Microsoft Azure IoT services.

WizFi360 in Various Ecosystems

DIY enthusiasts and hobbyists can easily develop their application for WizFi360 using WizFi360 Arduino library currently maintained by our friends in Kocoafab [2]. Furthermore, as WIZnet implemented Espressif-like AT commands, users may even use WiFiEsp library, for more details and guide refer to [3].

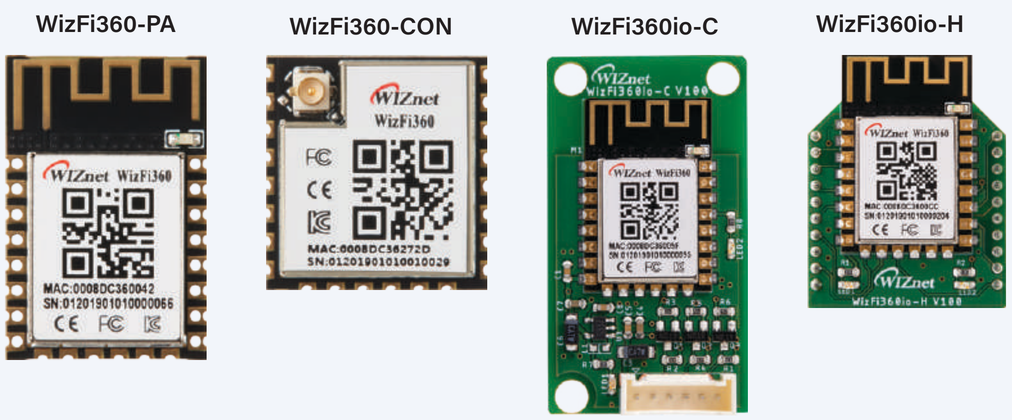
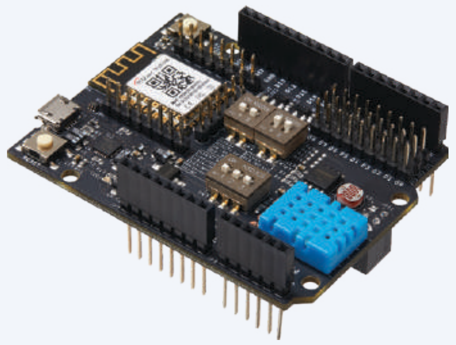
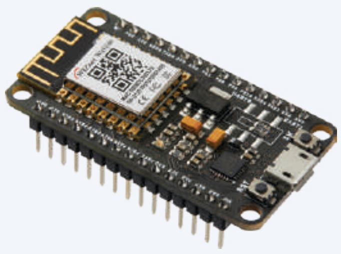


Figure 1: WizFi360 comes in two options: with onboard pattern antenna or u.fl connector. Also, WIZnet provides internet-offload (io) modules with various form factors.

WizFi360-EVB-Shield



WizFi360-EVB-Mini



WizFi360-EVB-Pico

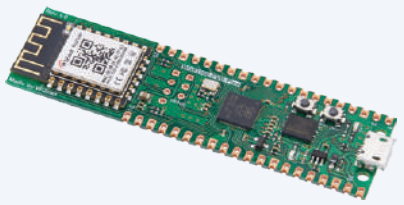


Figure 2: WIZnet provides various evaluation boards for easy project prototyping.

For RP2040 family boards, WIZnet already released C/C++ SDK and sample codes [4]. CircuitPython and MicroPython SDK are currently being prepared.

WIZnet joined ARM ecosystem in 2015, when WIZwiki-W7500 was certified by Arm Mbed. To continue and prosper partnership between two companies, WizFi360 was selected as one of the official shields for official Wi-Fi Shield on Arm Open-CMSIS-Pack and Keil Studio Cloud development.

We at WIZnet are trying to provide as many tools and example codes as possible to ease developers' work.

How to Procure?

Despite global silicon and semiconductor shortage, WIZnet is doing its best not to increase prices for their products. Unlike many manufactures, in March 2022 prices for WizFi360 were lowered up to 20% globally.

Furthermore, WIZnet launches new direct sales service [5] online storefront for VAR (value-added-resellers) partners and UCC (user-created-content) makers. WizFi360 is one of two products currently being sold through this service. It is necessary to fill out the contact form to get the most reasonable price and best lead time.

Interested in Testing WizFi360? Join WizFi360 Design Contest!

WIZnet is holding a WizFi360 design contest. To celebrate the launch of WizFi360-EVB-Pico, we are offering a free sample (WizFi360 module, WizFi360-EVB-Mini or WizFi360-EVB-Pico) to all participants. It's a great opportunity to bring your own ideas to life with a variety of provided open source and examples. Join our contest and win 1 of 30 iPad Pros. Follow the link [6] for more details.

To Conclude

In this article, we've taken a closer look at WizFi360 Wi-Fi module and how it is being used for development of various applications in different ecosystems. Since its release, WIZnet already sold more than 1 million modules globally.

WiFi module WizFi360 is a perfect choice for mobile wireless applications such as remote monitoring and sensor applications. Ease of integration and programming can vastly reduce development time and minimize system cost. Depending on your project needs, WIZnet is ready to offer firmware customization services. Should it be custom AT-command or application-specific program, we are ready to support all!

220277-01

Other WIZnet Wi-Fi Modules

Since 2015 WIZnet released a few Wi-Fi modules using chips from various manufacturers. In the current lineup there are:

- WizFi630S [7] - gateway module that integrates 1T1R 802.11 Wi-Fi, a 580 MHz CPU, Ethernet PHY, USB2.0 host, SD-XC, I2S, I2C and more GPIOs;
- WizFi310 [8] - small size wireless module for the highest level of integration, featuring 802.11b/g/n. This module is included in Ubidot supports boards.

Information about other modules can be found on WIZnet's website www.wiznet.io.

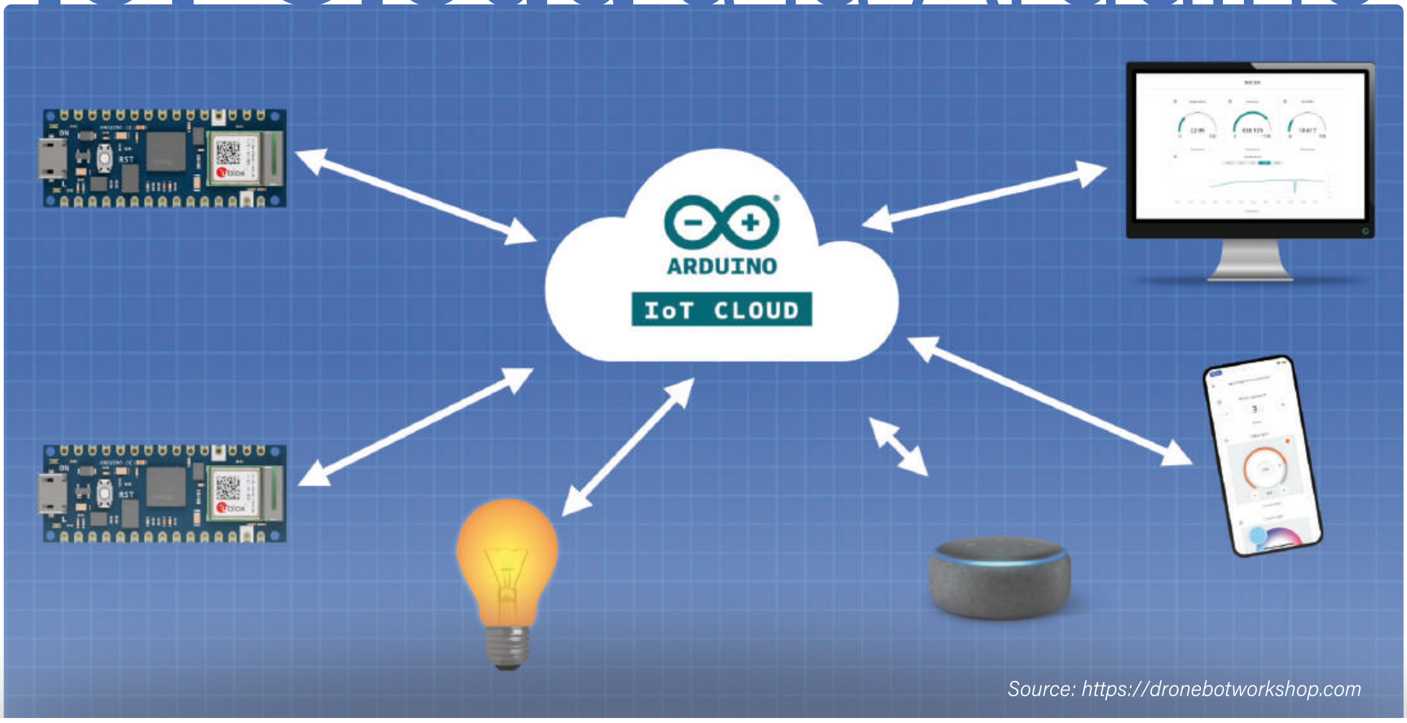
About WIZnet

WIZnet is a fabless IT company that provides Internet processors for the IoT.
WIZnet is the sole innovator to patent the hardwired TCP/IP technology into a microprocessor chip in 2001. Since then, a yearly average of 10 million WIZnet chips have been used in various embedded Internet devices worldwide.
Since establishing a loyal brand image with customers, WIZnet cooperates with 70 distributors worldwide with branch offices in USA, China, and India for competitive project marketing and faster technical support.

WEB LINKS

[1] WizFi360 datasheet: https://docs.wiznet.io/Product/Wi-Fi-Module/WizFi360/documents
[2] Arduino Library: www.arduino.cc/reference/en/libraries/wizfi360/
[3] WiFiEsp library: https://github.com/wizfi/WizFi360EVB-Arduino
[4] C/C++ SDK and sample codes for RP2040: https://docs.wiznet.io/Product/Open-Source-Hardware/wizfi360-evb-pico/
[5] Direct sales service: https://direct.wiznet.io
[6] Contest and free Sample: https://maker.wiznet.io
[7] WizFi630s: www.wiznet.io/product-item/wizfi630s
[8] WizFi310: www.wiznet.io/product-item/wizfi310

IoT Cloud a la Arduino



By Tam Hanna (Slovakia)

The Arduino IoT Cloud offers IoT application developers a convenient solution for implementing a cloud back end without having to struggle with MQTT. Curious? Let's take a look.

Lots of microcontroller applications nowadays involve an Internet of Things (IoT) application where information is disseminated via IoT cloud services and an MQTT broker. Creating this type of application using a local development environment like the traditional Arduino IDE can sometimes be awkward. The Arduino Cloud shifts the IDE up into the cloud so that your browser becomes a window into the IDE. We tried it out by sending a variable value to the cloud to make an LED flash on the bench. Then we tried to break it.

The basis of all IoT devices is of course the *Thing*. In the Arduino IoT Cloud development environment, the *Thing* is a virtual object that exists in the cloud. In the real world, it is as an object such as a server, a controller board, or a similarly "intelligent" device [1]. Here your *Thing* is built in the cloud using an online editor to write a Sketch describing how it should behave and respond by using a whole range of *Variables*.

Who Will Use the Arduino IoT Cloud?

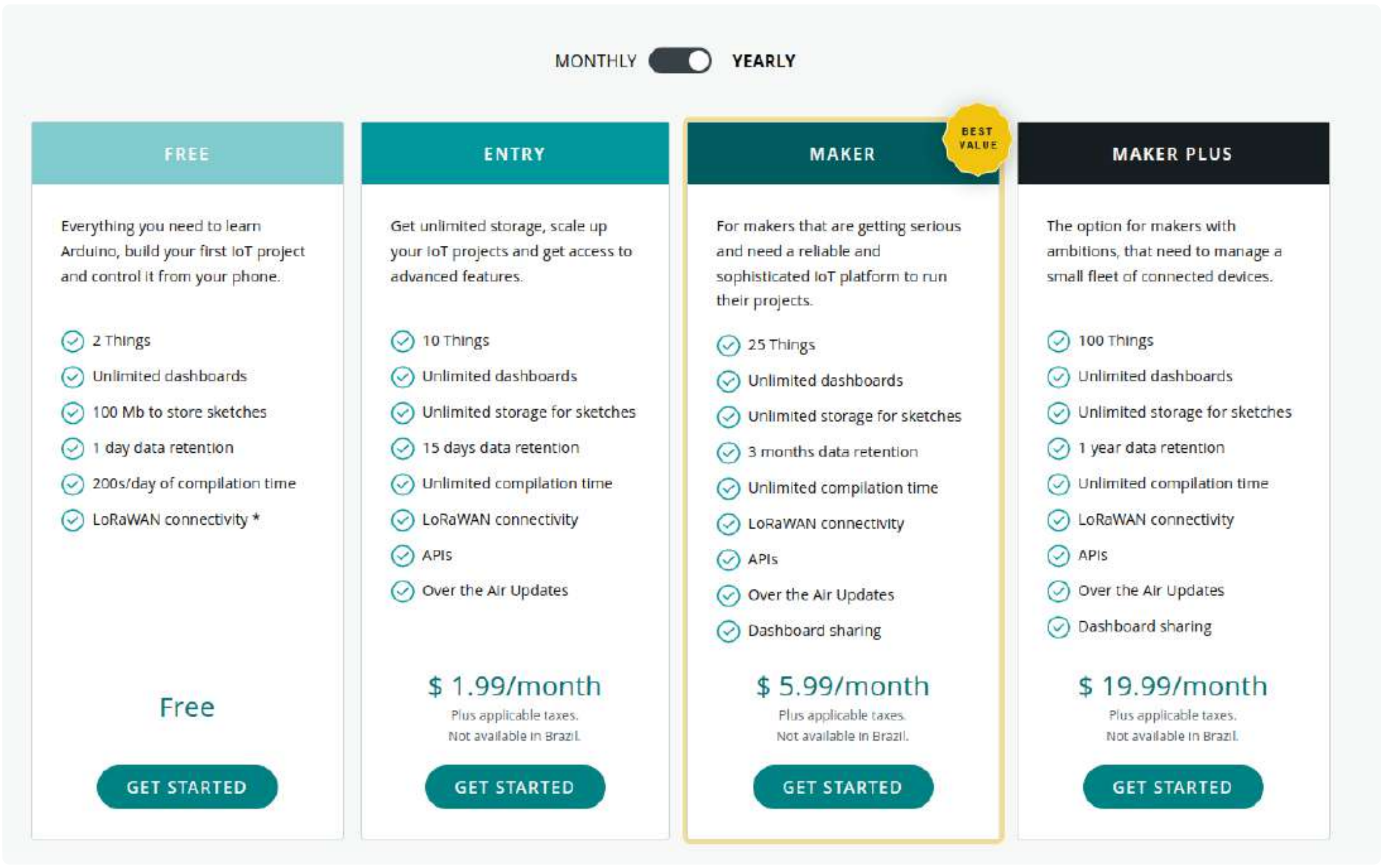
Before we begin, it's important to acknowledge that the Arduino IoT Cloud is not an alternative for other dedicated cloud computing platforms such as Amazon AWS IoT Core, Microsoft IoT Hub, or Yandex IoT Core. If you have large numbers of devices and data to manage, these more established IoT cloud services are the way to go.

At the introduction of the latest incarnation of The Arduino IoT cloud, Massimo Banzi, CTO at Arduino, expressed his ambitions for the platform saying that: "Arduino now offers a complete platform with the MKR family; providing a streamline way to create local IoT nodes and edge devices. These use a range of connectivity options and compatibility with third-party hardware, gateway and cloud systems. The Arduino IoT Cloud allows users to manage, configure and connect, not only Arduino hardware but also the vast majority of Linux-based devices — truly democratising IoT development."

Its support of the MKR IoT-targeted range of Arduino boards and some other popular third-party boards is a welcome addition and would make many IoT system developers give this accessible development platform a second look.

Setting Up the Hardware

Only the most basic plan of the four possible versions of the Arduino Cloud is free to use. You can check out the various plans and their features in **Figure 1** and decide which package best meets your needs. Of interest to the maker community generally is Arduino Cloud's support of third-party boards, such as the popular ESP8266 and ESP32 family of devices (**Table 1**), and the list of compatible platforms [2]. Driver libraries also allow various Linux-based systems to upload and download information to the Arduino cloud.


MONTHLY YEARLY

FREE

Everything you need to learn Arduino, build your first IoT project and control it from your phone.

- 2 Things
- Unlimited dashboards
- 100 Mb to store sketches
- 1 day data retention
- 200s/day of compilation time
- LoRaWAN connectivity *

Free

GET STARTED

ENTRY

Get unlimited storage, scale up your IoT projects and get access to advanced features.

- 10 Things
- Unlimited dashboards
- Unlimited storage for sketches
- 15 days data retention
- Unlimited compilation time
- LoRaWAN connectivity
- APIs
- Over the Air Updates

$ 1.99/month
Plus applicable taxes.
Not available in Brazil.

GET STARTED

MAKER BEST VALUE

For makers that are getting serious and need a reliable and sophisticated IoT platform to run their projects.

- 25 Things
- Unlimited dashboards
- Unlimited storage for sketches
- 3 months data retention
- Unlimited compilation time
- LoRaWAN connectivity
- APIs
- Over the Air Updates
- Dashboard sharing

$ 5.99/month
Plus applicable taxes.
Not available in Brazil.

GET STARTED

MAKER PLUS

The option for makers with ambitions, that need to manage a small fleet of connected devices.

- 100 Things
- Unlimited dashboards
- Unlimited storage for sketches
- 1 year data retention
- Unlimited compilation time
- LoRaWAN connectivity
- APIs
- Over the Air Updates
- Dashboard sharing

$ 19.99/month
Plus applicable taxes.
Not available in Brazil.

GET STARTED

Figure 1: Which plan is best for you? (Plan pricing as of 1/20/2022.)

An Arduino Nano RP2040 Connect module is used here as the target board to test the Arduino Cloud functions. This board is based on the RP2040 microcontroller from the Raspberry Pi Foundation and includes a u-blox Wi-Fi module. Before setting up the Arduino cloud, the board is connected to a computer via a USB cable.

First, we need to visit the website [3] and apply for a new Arduino account. For our purposes, we will stick to the basic free plan which is explicitly aimed at newcomers to the system. First, we click on the *Create Thing* button to create a new thing.

The overview in **Figure 2** now shows the configuration of the three basic components of the development environment. I have removed a previous Arduino account and all configuration parameters so that I start with a fresh device setup. The following steps are performed on a machine running Windows 10, but the process is identical on the Ubuntu Linux environment and I have found that hardware detection usually works better under Unix.

Now we can click on the shortcut icon in the *Device* section and choose to *Set Up an Arduino Device*. A few seconds after clicking this option, the back end will point out that the component called *Arduino Create Agent* is missing. Click the Download button to download the software and install it in the normal way.

Note that *Create Agent* is browser-specific: if you install under Chrome, you will need to reinstall. Any firewall warnings that pop up should be acknowledged. Make sure that you allow access to

Table 1: Boards supported by the Arduino Cloud

WLAN
- MKR 1000 WiFi
- MKR WiFi 1010
- Nano RP2040 Connect
- Nano 33 IoT
- Portenta H7

LoRaWAN
- MKR WAN 1300
- MKR WAN 1310

GSM/NB-IoT
- MKR GSM 1400
- MKR NB 1500

ESP32/ESP8266
- wide range of third-party boards

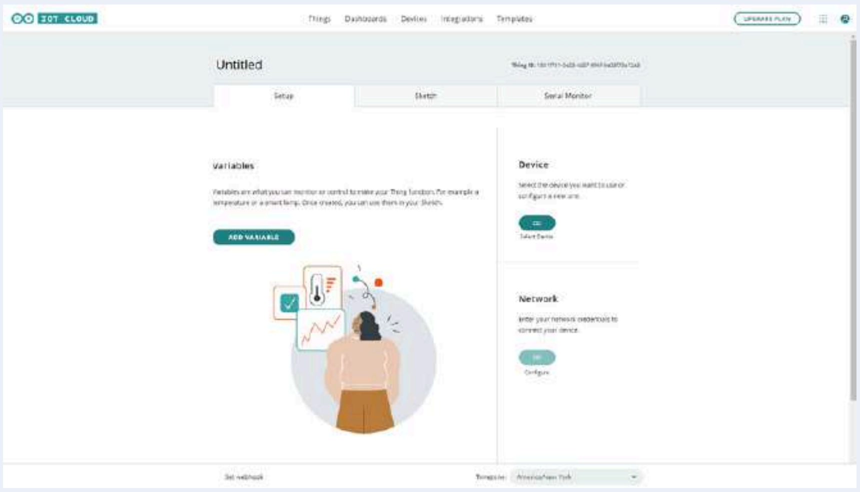
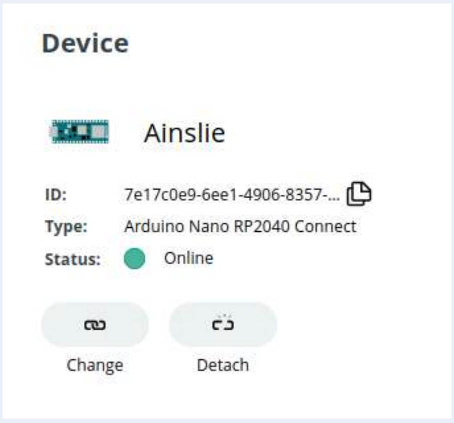


Figure 2: The IoT Cloud leads the developer step by step through to the goal.


Device

Ainslie

ID: 7e17c0e9-6ee1-4906-8357-...
Type: Arduino Nano RP2040 Connect
Status: ● Online

Figure 3: This Arduino is connected to the Cloud.

private and public networks alike. As a result, the *Arduino Create Agent* is now resident in your taskbar — in some cases it may need to be called up again from the start menu but that now completes the driver installation.

In the next step, we will refresh the view until the Arduino cloud informs us that our Nano RP2040 Connect board has been recognized. Next, click the *Configure* button to launch the configuration wizard — it will ask you to enter a 'friendly" name and then initialize the target system's secure element with the basic communications software.

Problems sometimes arise during network provisioning under Windows. The more reliable and successful method is to do this using Linux instead. Incidentally, the *Network* section is not automatically enabled by the Arduino cloud. It is only available when you create one of the variables intended for data exchange.

We can now click on *Variables*, to open a dialog and add a new variable. First we assign the name `ledIntenBool` and assign *Boolean* in the data type field. Funnily enough, the Arduino cloud not only supports C programming units, but also implements wrappers around real world variables.

If you want to limit yourself to C only, we recommend selecting the *Basic Types* option. Theoretically, we could then make settings in the *Variable Permission* and *Variable Update Policy* fields, but the default settings will be sufficient for our purposes, which is why we now close the dialog. In the next step, we then create a field of the *Integer Number* type, to which we give the name `ledIntenInt`.

Preparing the Code

After creating the variables, red dots in the Sketch tab, indicate changes to the program structure. Now it is possible to click on the shortcut icon in the *Network* section to enter the Wi-Fi settings. My preference is to enter the values using a command line tool such as *iwlist* on a Linux machine and then copy them to the clipboard.

In the next step, we switch to the *Sketch* tab and click on the *Verify and Upload* button. The Arduino Cloud then starts compiling the code and sends it to the connected RP2040 using the *Arduino Cloud Agent*. If the compiled code is delivered successfully, the message *"Untitled_dec25a uploaded successfully on board Arduino Nano RP2040 Connect (/dev/ttyACM0)"* is displayed.

Now after the obligatory Reset, the RP2040 will start phoning home using its Wi-Fi transmitter. After a while and pressing *F5* several times the device will appear with "*Status: online*" as shown in **Figure 3**.

If you chose to subscribe to one of the more comprehensive Arduino Cloud plans, you can receive software updates directly via Wi-Fi; but for our simple experiments using the free plan, a wired connection is necessary.

A Closer Look

The basic editor tool of the Sketch tab is not so useful, but by clicking on the *Open full editor* button, we get a fully-fledged cloud-based IDE that allows us to edit smaller projects more comfortably. First, let's look at the contents of the *thingProperties.h* file, which contains structural elements of the sketch.

First, we see the following declarations that provide the elements required for Wi-Fi access:

```
const char SSID[] = SECRET_SSID; // Network SSID (name)
const char PASS[] = SECRET_PASS; // Network password
// (use for WPA, or use as key for WEP)
```

The Arduino cloud takes care of entering the name and password using the settings already entered in the *Network* section. The next part contains the following two variables the names of which should be familiar from their declaration in the *Variables* section:

```
int ledIntenInt;
bool ledIntenBool;
```

The Arduino cloud implements the variables "in the back end" as standard C variables equipped with additional properties. These properties can be found, among other things, in the `initProperties`

Method, which takes care of setting up the primitives and structures required for cloud communication according to the following scheme:

```
void initProperties() {
ArduinoCloud.setThingId(THING_ID);
ArduinoCloud.addProperty(ledIntenInt, READWRITE,
ON_CHANGE, onLedIntenIntChange);
ArduinoCloud.addProperty(ledIntenBool, READWRITE,
  ON_CHANGE, onLedIntenBoolChange);
}
```

It is interesting here that the `addProperty` method, takes care of "registering" the attribute. Note the passing of the `onLedIntenIntChange` and `onLedIntenBoolChange` function pointers - they will play an important role later on.

The application control function is described in the sketch, which begins with the inclusion of the header (not shown here). This is then followed by the sketch initialization, which runs according to the following scheme:

```
void setup() {
 Serial.begin(9600);
 delay(1500);

 initProperties();

 ArduinoCloud.begin(ArduinoIoTPreferredConnection);
 setDebugMessageLevel(2);
 ArduinoCloud.printDebugInfo();
}
```

From the point of view of the Arduino programming environment, the Arduino cloud is a hardware driver like any other. The `ArduinoCloud` global object exposes a set of functions that your code uses to communicate with the cloud driver. Of particular importance here is the call to `setDebugMessageLevel`, which sets the "verbosity" of the driver — the higher the value, the more debug information the cloud driver outputs over the board's serial port.

In the case of "complicated" drivers, the question always arises as to how the processing power is allocated. The project skeleton created for us by the Arduino Cloud can answer this question in the loop method, which takes care of the allocation of computing power according to the following scheme:

```
void loop() {
 ArduinoCloud.update();
}
```

The Arduino cloud gives us the following three listener methods by default:

```
void onTestScheduleChange() {
}
void onLedIntenBoolChange() {
}
void onLedIntenIntChange() {
}
```

`onLedIntenBoolChange` and `onLedIntenIntChange` are responsible for the variables created remotely in the back end, while `onTestScheduleChange` helps implement "internal" functions of the Arduino cloud.

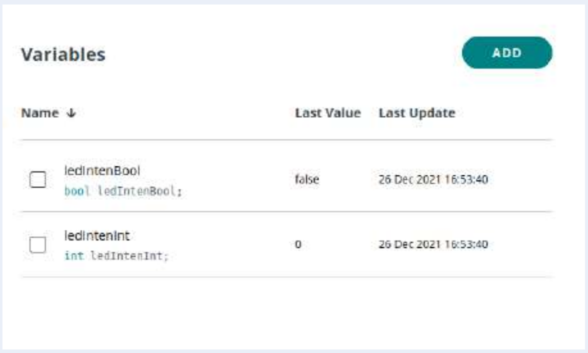
In the next step, we can take care of this by making use of the built-in "intelligent" capabilities of the Cloud. The Arduino board we are using here is fitted with a standard (red) LED on Pin 13 and an RGB LED (channels LEDR, LEDG and LEDB), which can be driven by three PWM signals to change the overall colour output.

Now we can return to the `setup` function, and initialise the necessary pins:

```
void setup() {
. . .
 ArduinoCloud.printDebugInfo();

 pinMode(LED_BUILTIN, OUTPUT);
 pinMode(LEDB, OUTPUT);
}
```

Changes triggered in the cloud activate the listener, which writes the incoming values to the hardware:

```
void onLedIntenBoolChange() {
 digitalWrite(LED_BUILTIN, ledIntenBool);
}
void onLedIntenIntChange() {
 analogWrite(LEDB, ledIntenInt);
}
```



Name ↓	Last Value	Last Update
ledIntenBool `bool ledIntenBool;`	false	26 Dec 2021 16:53:40
ledIntenInt `int ledIntenInt;`	0	26 Dec 2021 16:53:40

Figure 4: The Arduino IoT cloud provides detailed information about the current state of the information stored in a variable.

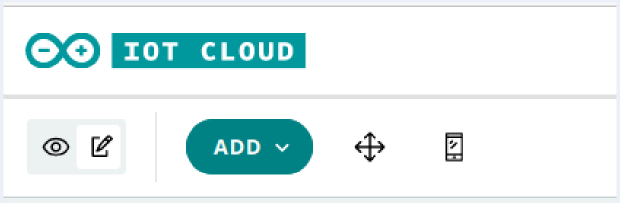
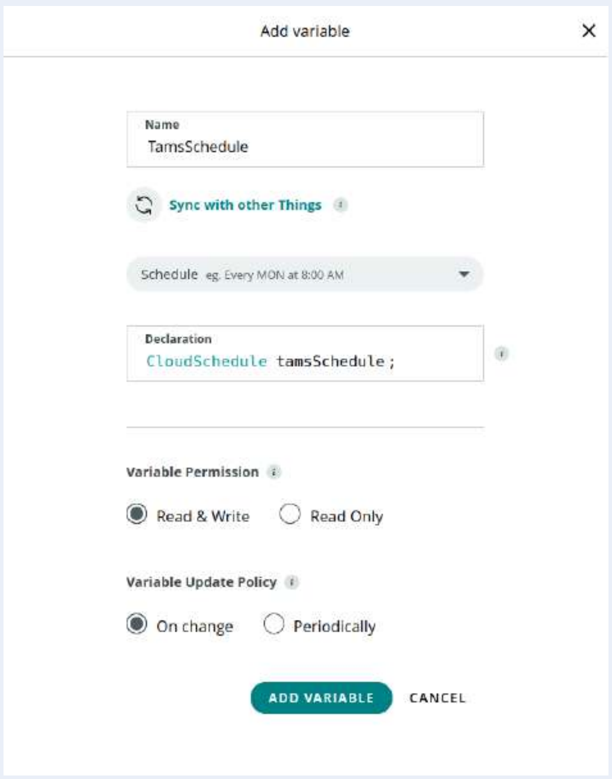


Figure 5: Switch the dashboard to edit mode by clicking on the pencil icon top left.

Figure 6: The editing interface for controls is "modal".



Add variable

Name: TamsSchedule

Sync with other Things

Schedule eg. Every MON at 8:00 AM

Declaration: `CloudSchedule tamsSchedule;`

Variable Permission: [✓ Read & Write] Read Only

Variable Update Policy: [✓ On change] Periodically

ADD VARIABLE CANCEL

Figure 7: The Cloud Scheduler is a data type just like all the others.

At this point, you can transfer the sketch to the circuit board again. The RGB LED uses common anode connections so individual colours are controlled by outputting a 0 to turn on the corresponding LED. The variables are initialised as shown in **Figure 4**, so that the blue diode of the RGB LED lights up after successful initialization.

Modifing the Variable Contents

If we return to the main back end of the Arduino cloud, we can click on the *Dashboards* tab, which, if this is a new account, will prompt you to create a new dashboard. Now click the *Build Dashboard* button to launch the editor, which will take a moment or two even if you have a fast Internet connection.

To modify and add elements to the dashboard click on the edit (pencil) icon on the top left of the display (**Figure 5).** Once this mode is selected the blue *ADD* button appears which you can use to show the drop down list of available widgets. First, we select a *Switch* widget, which then appears in the editing interface shown in **Figure 6**.

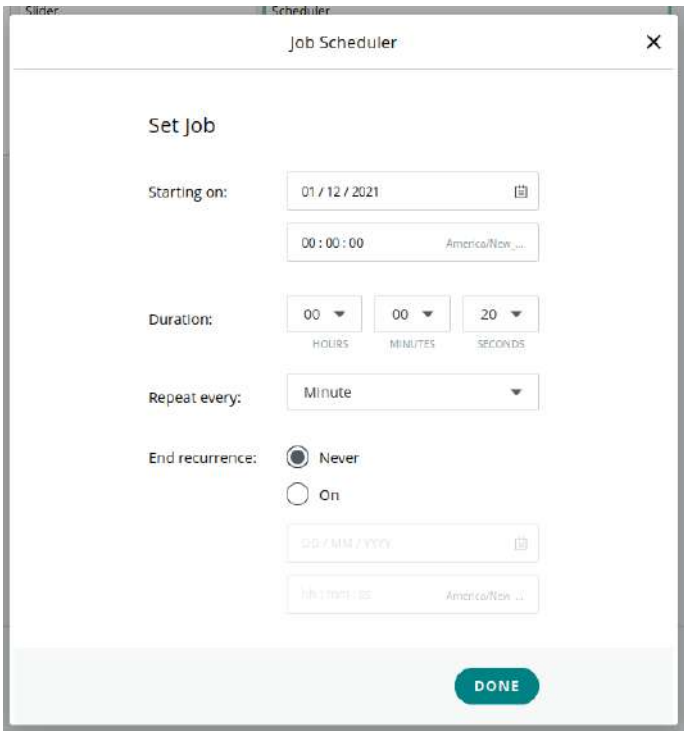
Now over to the right of the display is the *Linked Variable* field with a link button. Click on it to activate a list of all Things and Variables contained in the cloud account. Here we can choose the `ledIntenBool` variable and link it using the *Link Variable* button. The state of `ledIntenBool` will now be controlled by the state of the Switch. A click on the DONE button closes the editing interface and the switch is now incorporated into the dashboard. Now we can click on the eye icon to release the switch to activate. Toggling the switch on and off now controls the red LED next to the microUSB socket.

In order to be able to set the brightness of the blue LED, we have to switch the dashboard editor back to edit mode and add a new control again by using *Add -> Widgets*. This time I chose the *Slider* type. In its editing interface, we set its *Value Range* from 0 - 255. The link is made by the variable `letIntenInt`, which represents the RGB LED "brightness control". Last but not least, we switch to activation mode here and see that changes to the slider position now affect the brightness of the blue LED.

Using the Scheduler

As I write this article, a function for setting up scheduled tasks or web cron-jobs is a new feature: it uses a variable type called *CloudSchedule* that you can define to be true or false at a specific time and for a specific time period. It's not necessary to invoke any timer function because this variable is set or reset automatically in the Arduino IoT Cloud, according to how you configure it. Tasks can then be triggered by checking the state of this variable. To demonstrate the possibilities, let's create a new variable of type *Schedule*. **Figure 7** shows the desired configuration. We can now see the new variable type in the code:

```
CloudSchedule tamsSchedule;
```



Set Job

Starting on: 01 / 12 / 2021 — 00 : 00 : 00 America/New ...

Duration: 00 HOURS, 00 MINUTES, 20 SECONDS

Repeat every: Minute

End recurrence: [✓ Never] On

DONE

Figure 8: The Scheduler is configured using the Dashboard.

The time parameters for this variable are configured via the dashboard. The settings page shown in **Figure 8** show the control elements provided for this purpose.

In the following step, we need to take care of the "local" processing of the values contained in `tamsSchedule`:

```
void setup() {
. . .

 pinMode(LED_BUILTIN, OUTPUT);
 pinMode(LEDB, OUTPUT);
 pinMode(LEDR, OUTPUT);
}

void loop() {
 ArduinoCloud.update();
 // Your code here
 if(tamsSchedule.isActive()){
  digitalWrite(LEDR, HIGH);
 }
 else{
  digitalWrite(LEDR, LOW);
 }
}
```

It is important here that the "writing out" of the information supplied in `tamsSchedule` is the exclusive task of the developer. The cloud is just limited to periodically updating the value contained in `tamsSchedule`. The continual polling procedure shown here in the `loop` structure may not be optimal from a resource point of view, but it works without problem. The program can now be sent to the Arduino where you can watch for the periodic red flashes coming from the RGB LED.

At this point, I could not resist another small system test to see what happens when the RF link goes down. The "lab Wi-Fi" was turned off and the Arduino began to act erratically by random switching the blue element of the RGB LED and also the LED on pin 13, eventually after a few seconds it performed a complete restart.

At the time this article went to press, it was not really clear to me how exactly the Arduino cloud recovers from the loss of the radio link between the end device and server.

A Convenient Option

It's clear that the Arduino IoT Cloud is still a work in progress and is under constant development and improvement. It does however have great potential and offers the IoT application developer a convenient and low-threshold option for implementing a cloud back end without having the need to struggle with MQTT and Co. Despite the odd hiccup I can thoroughly recommend this product!

210550-01

Contributors

Idea, Illustrations and Text: Tam Hanna
Editor: Rolf Gerstendorf
Translation: Martin Cooke
Layout: Harmen Heida

Questions or Comments?

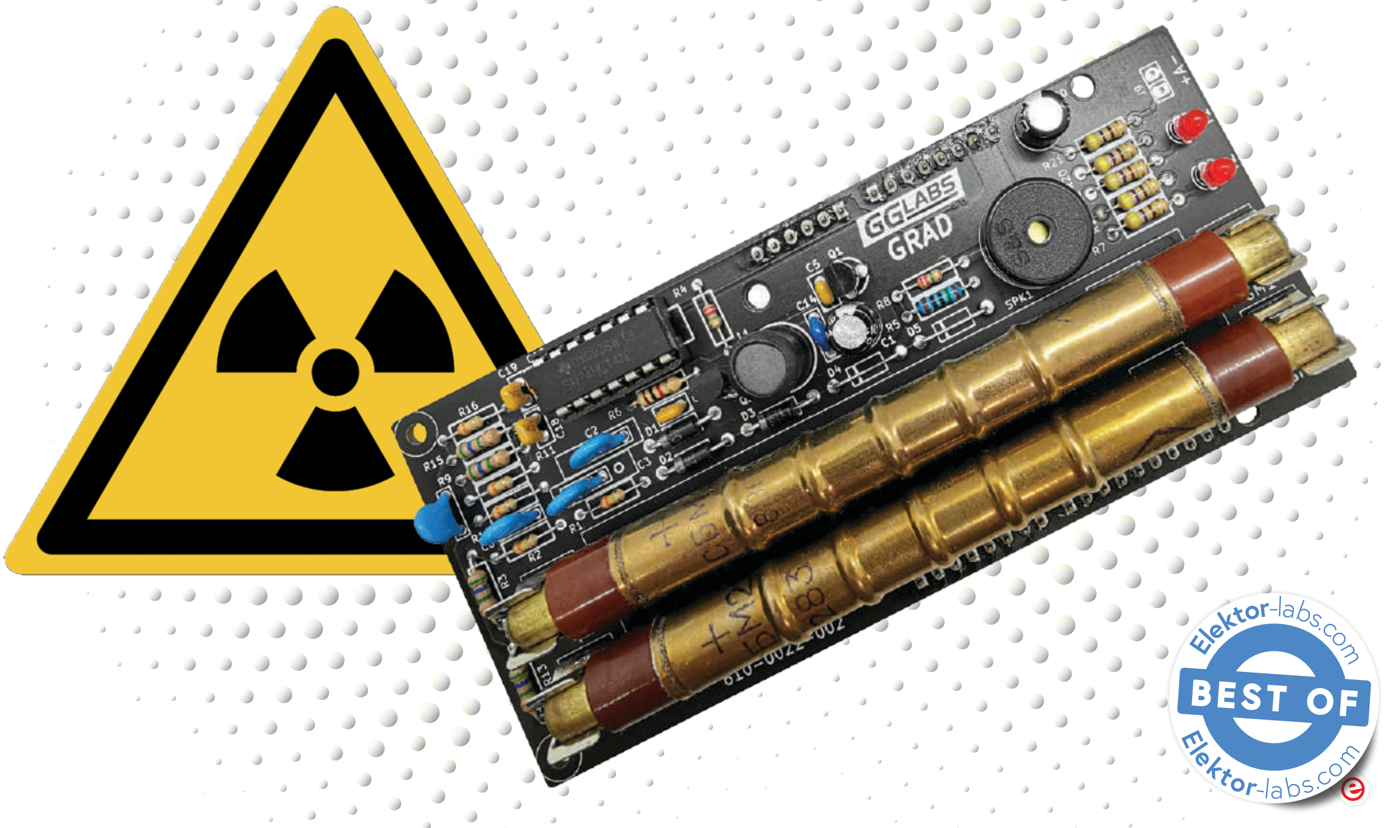
Do you have any technical questions or comments about this article? Contact the author at tamhan@tamoggemon.com or the Elektor team at editor@elektor.com.

WEB LINKS

[1] Digital twin: https://en.wikipedia.org/wiki/Digital_twin
[2] Boards supported by the Arduino Cloud: https://bit.ly/3t8Vl3W
[3] Arduino Things: https://create.arduino.cc/iot/things

Dual Geiger-Müller Tube Arduino Shield

A High Sensitivity, Very Low-Power Radiation Sensor



By Gabriele Gorla (Italy/USA)

With two tubes for increased sensitivity, this Geiger-Müller tube shield can turn your Arduino Uno into an instrument for measuring and recording nuclear radiation. It can even be combined with a Dragino shield for LoRa-connectivity. Collect radiation data in the "field" — and get access to it from all over the world!

Like many other hobbyists, I have always been fascinated by radioactivity and the sensors to detect it. Geiger-Müller tubes [1] are a common and relatively inexpensive way to measure radiation. My "GRAD" project is a complete solution for radiation counting in an Arduino shield form factor. Its main features are dual tube support to increase sensitivity and very low power consumption.

Introduction to Geiger-Müller Counters

Every Geiger-Müller counter requires four essential functional blocks.

- **Geiger-Müller Tube**. The tube has two terminals and is filled with a low-pressure gas mixture. When biased with the appropriate voltage the gas will ionize and briefly conduct electricity every time it is hit by radiation. Depending on the tube type it is possible to detect alpha, beta and gamma particles. The SBM-20 used in the GRAD is sensitive to gamma and high energy beta radiation.
- **High-Voltage Power Supply.** The tube must be operated in the so-called Geiger Plateau. This is a region where the pulse count is

almost independent of the bias voltage. For common tubes the bias voltage is between 400 V and 500 V. The SBM-20 optimal point is around 400 V.

- **Pulse Detector.** The pulses out of the tube are very short and of variable voltage. The pulse detector conditions the signal so it can be easily counted by the following stage.
- **Pulse counting.** Pulses are counted over a fixed time interval to calculate a CPS (counts per second) or CPM (counts per minute) value. This can be roughly converted to a dose rate by using the parameter on the tube datasheet. An Arduino board is used for pulse counting and its visualization and/or logging.

Power Supply

There are many circuits on the web for Geiger-Müller tube power supplies. Many are boost converters built around the 555 timer, some open loop, some with feedback. The open-loop designs were not considered as they require tuning per board and do not provide a stable voltage at high pulse counts. Closed-loop designs are more suitable as they provide the required stability. However, to keep overall power consumption low, special attention should be paid to the feedback loop (12 μA of current @ 400 V is ~5 mW).

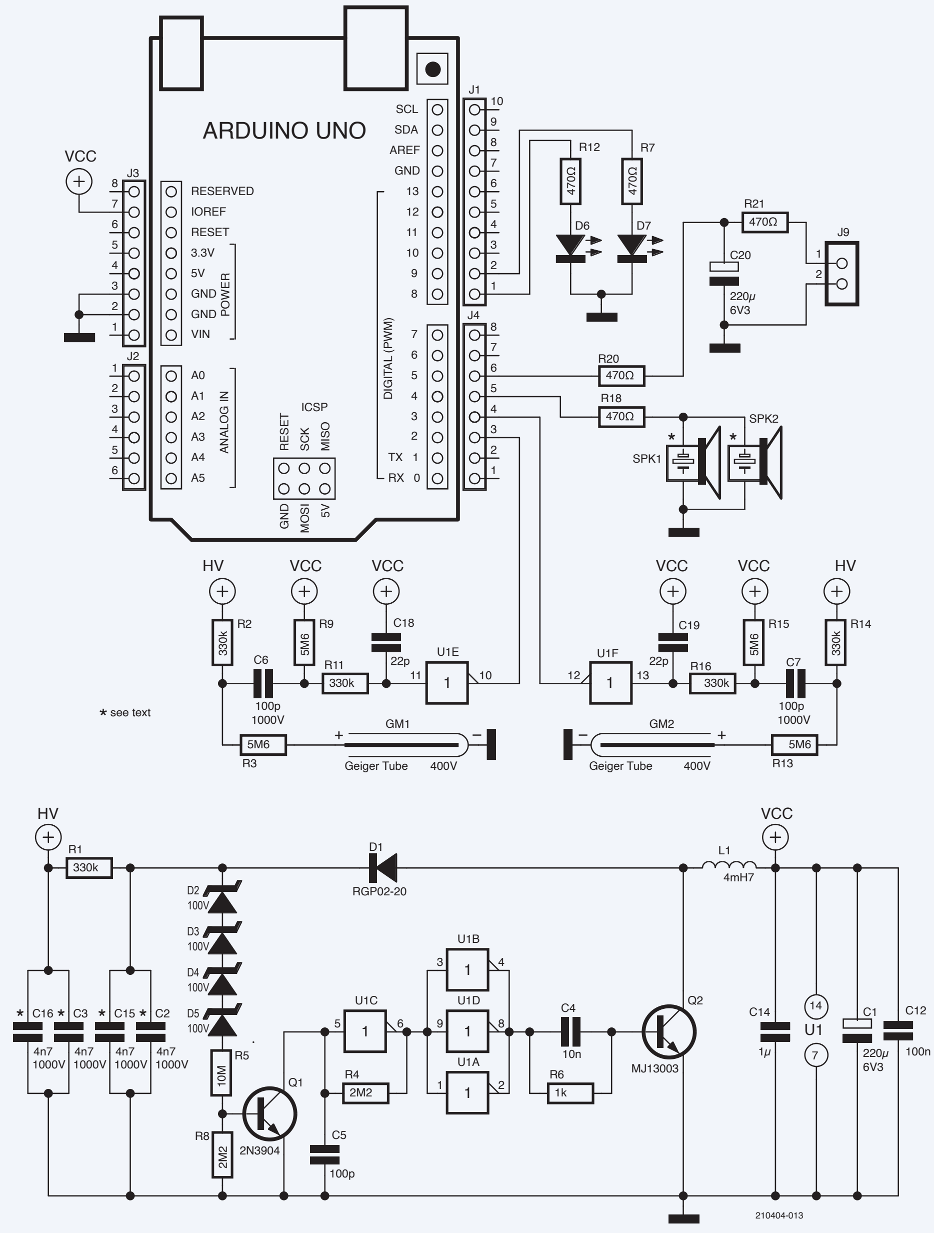


Figure 1: Schematic diagram of the GRAD03 project [3].

To avoid the power penalty of high-voltage feedback, the most elegant solutions use a non-isolated step-up transformer like the Analog Devices LT3420. It detects the voltage on the primary side.

We implemented a simple switch mode boost converter with a very low current feedback. Our design — shown in **Figure 1** — is heavily based on the Theremino Geiger adapters [2]. We mixed the SMD, DIY and "Flintstones" versions in a fully PTH design with Zener diode feedback.

The Schmitt trigger inverter U1C, R4 and C5 form an oscillator that generates the pulses to drive the main switch Q2. U1A, U1B and U1D are in parallel to increase Q2 base current.
The feedback is implemented with a series of very low leakage current Zener diodes (D2 to D5). When the output voltage exceeds the total Zener voltage, Q1 turns on and stops the oscillator. When the output voltage drops again Q1 will turn off releasing the oscillator. R1 and C3 further reduce the output ripple. Additional series resistors feed each tube separately (R2 and R3 for GM1, R13 and R14 for GM2).

Counting

Pulses are picked up from the high voltage side of the tube using a DC blocking capacitor (C6 and C7). The pulse is shaped by the remaining Schmitt trigger inverters (see **Figure 2**).

J1 to J4 are the standard Arduino connectors, but GRAD only uses a few pins. Digital inputs D2 and D3 were chosen as they are interrupt pins on the Arduino Uno. This allow the Arduino to count pulses in the background while it is performing other actions.

Arduino digital outputs D8 and D9 are connected to a pair of LEDs

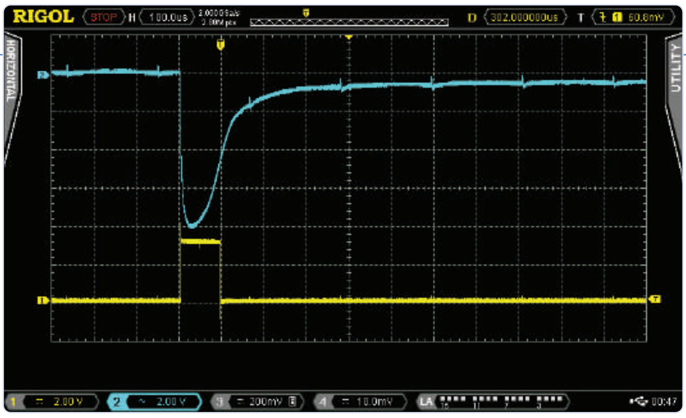


Figure 2: Scope capture of ionizing radiation hitting tube and pulse shaping.

to visually show when ionizing radiation hits the respective tube. Pin D4 is connected to piezo speaker SPK1 to provide audible feedback.

Finally, digital output D5 is connected to a low-pass filter formed by R20, R21 and C20 to connect a simple analog 10 mA panel meter (J9).

PCB Design

The schematic and PCB (**Figure 3**) were designed in KiCad. The design files are available for download at this project's Elektor Labs page [3] in the Project's Elements section. The PCB's gerber and drill files can be found there too. You can use these to order a PCB from your preferred supplier. Last but not least, you'll find an Excel sheet there with a very detailed Bill of Materials, especially documenting the more critical (high-voltage!) components needed for this project, including data on manufacturers, type numbers and even order codes.

Figure 4 shows the assembled PCB with SBM-20 tubes installed. Note that component pairs SPK1/SPK2, C3/C16, and C2/C15 respectively just provide alternative footprints on the PCB; so only SPK1 or SPK2 will be installed. The same goes for C3 or C16, and C2 or C15. J5 and J6 on the PCB denote two add extra footprints for connecting the negative terminals of shorter Geiger tubes.

Tube Options

The board is designed to work with 105-mm Soviet SBM-20, STS-5 and the Chinese J305 or the 90-mm J305 and M4011. Any other 400-V tube will also work with some customized tube mounting. In case of custom mounting, to minimize parasitic capacitance, it is important to keep the positive wire as short as possible. For tubes requiring different voltages, the Zener diode(s) should be changed to obtain the required voltage. Any diode with 0.5 μA or less of leakage should work.

Performance

The power supply consumes 325 μW (65 μA) at 5 V. In battery-operated devices the board can also be powered with 3.3 V. At this lower voltage, the power consumption drops to 150 μW (45 μA).

Arduino Sample Code

A simple Arduino sketch to drive the counter is available in the Project

COMPONENT LIST

Resistors
R1,R2,R11,R14,R16 = 330 k
R3,R9,R13,R15 = 5.6 M
R4,R8 = 2.2 M
R5 = 10 M
R6 = 1 k
R7,R12,R18,R20,R21 = 470 Ω

Capacitors
C1,C20 = 220 μF, 6.3 V
C2,C3 = 4700 pF, 1000 V
C4 = 10 nF, 6.3 V
C5 = 100 pF, 50 V
C6,C7 = 100 pF, 1000 V
C12 = 100 nF, 50 V
C14 = 1 μF, 6.3 V
C15,C16 = 4700 pF, 1000 V
C18,C19 = 22 pF, 50 V

Inductors
L1 = 4.7 mH

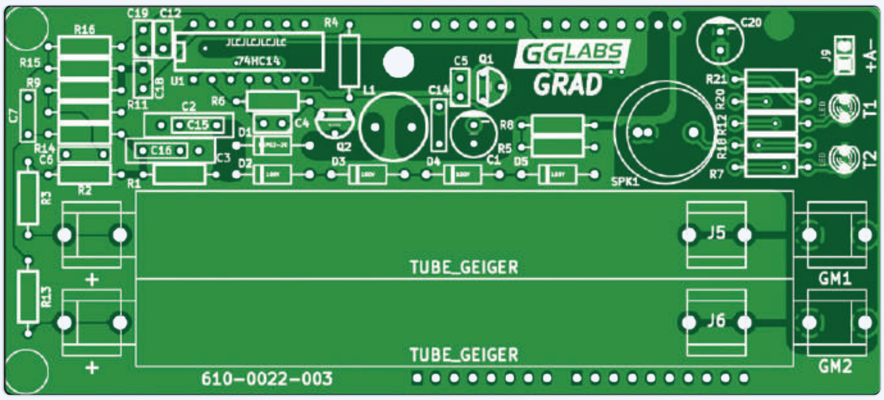


Figure 3: The PCB layout.

Semiconductors
D1 = diode 1 A, 800 V
D2,D3,D4,D5 = Zener diode 100 V 1.5 W
D6,D7 = red LED 3 mm (Marked T1 and T2 on PCB)
Q1 = 2N3904
Q2 = MJ13003
U1 = 74HC14

Miscellaneous
GM1,GM2 = Geiger-Müller tube, e.g. SBM-20
J1,J2,J3,J4 = set of Arduino Uno shield connectors
SPK1 = AC piezo buzzer, e.g. AC-1205G-N1LF

Elements section of the Elektor Labs page. Every 60 seconds, the software will output via the serial port: a comma-separated values (CSV) row with a sequential number, raw count for each tube, a moving average of `cnt1 + cnt2` and a μSv/h dose rate based on the average count. The following output data fragment shows the start of a background radiation measurement made in Santa Clara, CA, using GRAD with two SBM-20 tubes.

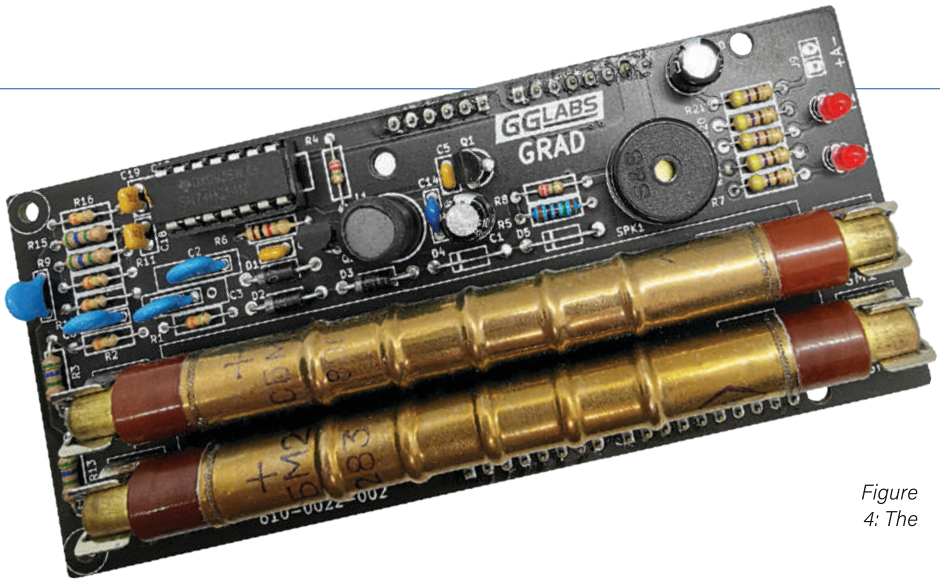


Figure 4: The

```
Seq , cnt1 , cnt2 , avg10 , μSv/h
1   , 14   , 19   , 33.0  , 0.075
2   , 10   , 17   , 32.3  , 0.073
3   , 16   , 12   , 31.8  , 0.072
4   , 16   , 24   , 32.5  , 0.074
5   , 13   , 11   , 31.6  , 0.072
```

The graph in **Figure 5** shows the complete measurement over a longer period of time.

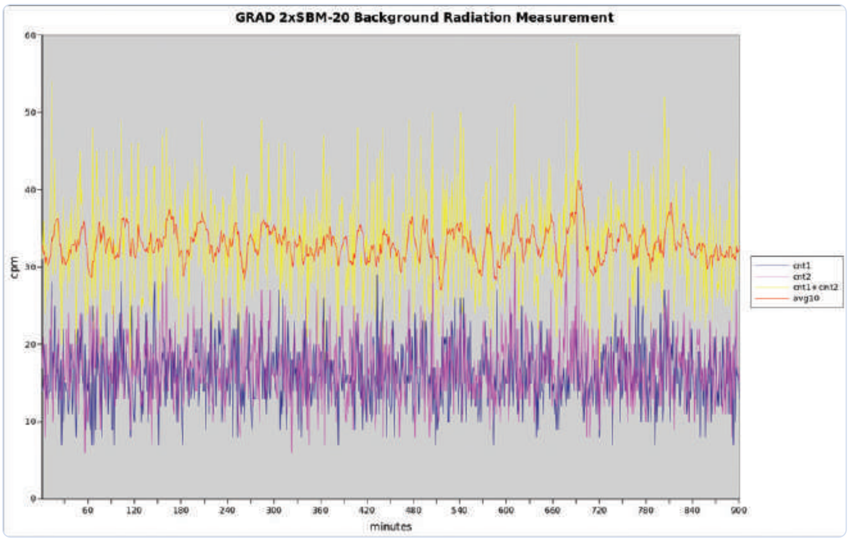


Figure 5: Background radiation in Santa Clara, CA.

There are a few defines at the beginning that are configurable to match the board configuration and/or user preferences. The first set configures the Geiger-Müller tube parameters and the moving window size for the reporting.

```
#define CPM2USV 220 // tube CPM to
                    // μSv/h conversion
                    // factor
#define TUBES 2     // number of tubes
                    // installed
#define WSIZE 10    // moving average
   window (in
                    // minutes)
```

The `CPM2USV` parameter is the count per minute for 1 μSv/h for your specific tube. Unfortunately, there is no published "correct" number for this parameter. For the SBM-20 radiation, enthusiasts on the web use values that range from 130 to 220 (conversion factor of 0.0075 to 0.0045).

The `TUBES` parameter, as the name states, defines the number of tubes installed. Valid values are 1 and 2. Finally, the `WSIZE` define the moving average window size for the counts. The default value of 10 defines a 10-minute window.

The second set of parameters defines how the optional hardware functions are connected:

```
#define LED1_PIN     8  // pin for TUBE1 LED
#define LED2_PIN     9  // pin for TUBE2 LED
#define SPKR_PIN     4  // pin for speaker connection
#define LED_BLINK_MS 20 // duration of LED blink for
                        // each count
```

All three `...PIN` defines select the Arduino pin where the LED or the speaker are connected. The `LED_BLINK_MS` defines the time that the LED will stay on for each count that the tube receives.

LoRaWAN Networking

The basic sketch requires the unit to be tethered to a computer through USB. Often it is desirable to place the sensor far from the computer, where a USB connection is not available.

A Dragino LoRa shield can be stacked to enable LoRa connectivity. Unfortunately, the Dragino shield also uses Arduino pin D2, so there are two options — a single-tube option and a two-tube option.

Single-tube option: In this case, only tube GM2 can be used and pin 10 of Schmitt trigger U1 must be disconnected to prevent interference with the LoRa communication. The pictures in **Figure 6** show the full stack with a single soviet SBM-19.

Two-tube option: In this case, both the Dragino shield and the GRAD

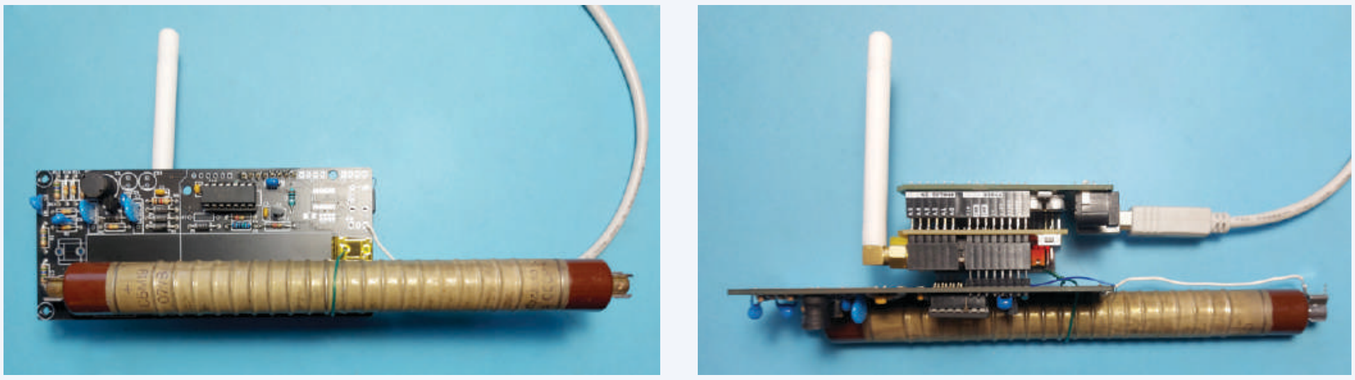


Figure 6: GRAD03 board, Arduino Uno and the Dragino LoRa shield combined.

board must be modified. For the Dragino shield, R5 and J_DIO0 must be removed. A wire should be used to connect the radio DIO0 to Arduino D7.

For the GRAD board R7, R12, C20 and the two LEDs should be removed. R21 should be replaced with a short, and a single LED should be placed at the J9 site (replace the analog meter output).

A sketch that communicates to the The Things Network (TTN) and posts tube counts every 60 seconds is available for download. It uses the LMIC Arduino library to drive a Dragino LoRa shield and it will connect to The Things Network using Over The Air Activation (OTAA). A simple Node-RED flow is used to display the data (see **Figure 7**).

Measuring Radiation

To verify the correct function of the Geiger counter a radiation source is required. These can be purchased online from special suppliers. Alternatively, specific vintage items that were built using small quantities of radioactive materials can be purchased at a thrift store or Ebay. Common radioactive items include: uranium glass items, thorium lantern mantles and selected colors of vintage fiestaware items.

Lacking any of these, another simple way is to check the decay product of Radon captured by the air filter from an air conditioner or an air purifier (**Figure 8**). These have a relatively short half life (in the order of tens of minutes) so make sure you run the AC for a couple of hours and then measure the filter immediately. If the filter is fine enough to it will emit radiations at a rate many times higher than background in your area.

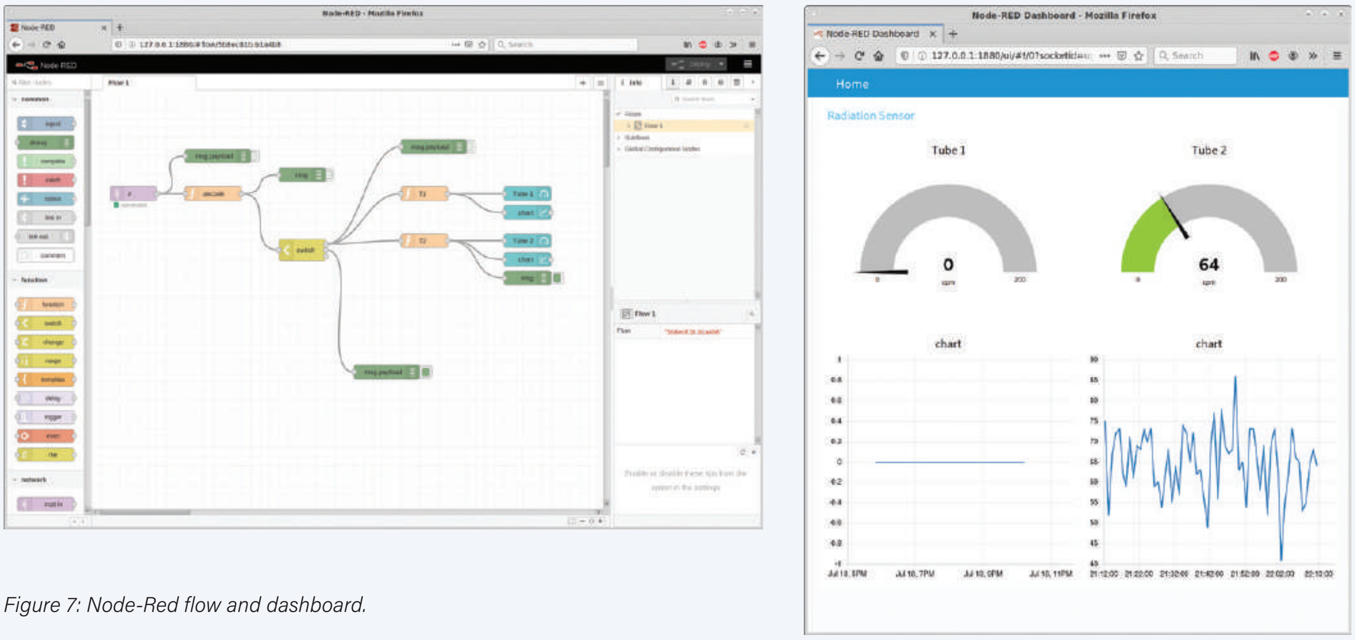


Figure 7: Node-Red flow and dashboard.

The chart below shows the count reaching almost ten times the background and then following the exponential curve typical of radioactive decay.

210404-01

This article is based on the material presented on the Elektor Labs page of this project [3]. There you'll find all the downloads for GRAD03, plus discussions and remarks on this subject.

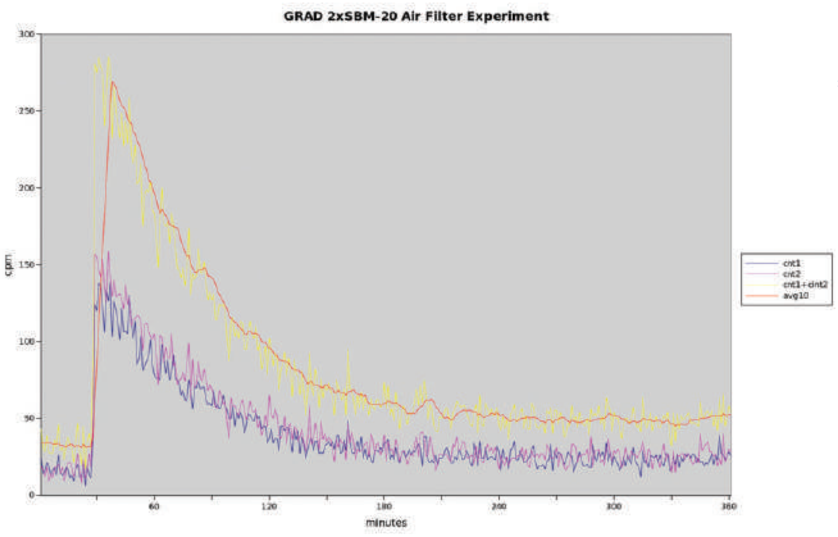


Figure 8: Radiation from an air conditioner filter due to radon decay.

Contributors

Design: **Gabriele Gorla**
Text: **Gabriele Gorla, Luc Lemmens**
Illustrations: **Gabriele Gorla, Patrick Wielders**
Editors: **Jens Nickel, C. J. Abate**
Layout: **Harmen Heida**

Questions or Comments?

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

RELATED PRODUCTS

- **Arduino Uno SMD Rev3 (SKU 19938)**
  www.elektor.com/19938
- **MightyOhm Geiger Counter Kit (incl. Case) (SKU 18509)**
  www.elektor.com/18509

WEB LINKS

[1] Geiger-Müller tubes: https://en.wikipedia.org/wiki/Geiger%E2%80%93M%C3%BCller_tube
[2] Theremino Geiger adapter (pls see three last designs on this page): www.theremino.com/en/technical/schematics
[3] This project on Elektor Labs: https://bit.ly/3giMntz

Advertisement

DISTRELEC
QUALITY, EXPERTISE AND RELIABILITY:
Explore cost-effective solutions from your local distribution expert
embeddedworld2022
Exhibition&Conference
... it's a smarter world
Visit us: Hall 2 - Booth 2-248

MonkMakes Air Quality Kit for Raspberry Pi

Measures Temperature and eCO_2

AIR QUALITY
for Raspberry Pi
Download the Instructions by scanning the QR code or visit:
monkmakes.com/downloads/pi_aq

By Luc Lemmens (Elektor)

Most of us are now stuck in (private) rooms, so modules for measuring the air quality cheaply are gaining popularity. The MonkMakes Air Quality Kit measures equivalent CO_2 and temperature. It is especially designed to be used with a Raspberry Pi 400, but it can also be connected to other Raspberry Pi models using the jumper wires and an included GPIO template.

Thermometers have long been used to monitor room temperature, and in recent years, (e)CO_2 meters have become increasingly popular for monitoring air quality. Too much carbon dioxide (CO_2) has a negative effect on concentration, and at even higher levels it is bad for your health. This kit measures the quality of the air in a room (how stale the air is) as well as the temperature. It is meant as an add-on for the Raspberry Pi, but it also can be used as a stand-alone device. The board has a buzzer and a bar of six LEDs (two green, two orange, and two red) that displays the air quality. Temperature and air quality readings can be processed by a Raspberry Pi. The buzzer and LED display can be controlled by the host.

The kit comes without printed documentation, but it provides a link to the MonkMakes website where the datasheet and instructions can be downloaded [1]. These documents contain all relevant information; they help the user to connect and use the board. Example applications in Python are available for download on Github [2].

The Hardware

Besides the six indicator LEDs and the buzzer — the big square component in the middle of the PCB in **Figure 1** — the board contains a power

LED, a temperature sensor, an eCO_2 sensor, a microcontroller and of course a 40-pin connector that fits directly on the expansion connector of a Raspberry Pi 400 (**Figure 2**). Perhaps needless to say: other Raspberry Pi boards can not be connected directly, for that the jumper wires are included. The four connections needed (two for the power supply, two for the serial connection) are shown in the print on the MonkMakes board and on the supplied template to match the corresponding pins of the GPIO connector of the Raspberry Pi, as illustrated in **Figure 3.** The power LED lights up as soon as the 3.3 V supply voltage is switched on, and so will one of the eCO_2 level LEDs.

The temperature sensor is a Texas Instruments TMP235 [3]. Its output voltage is proportional to the temperature. For CO_2 measurement, the MonkMakes board uses a CCS811 TVOC (Total Volatile Organic Compounds) sensor [4]. This does not actually measure the level of CO_2, but rather the level of a group of gasses called Volatile Organic Compounds (VOCs). When indoors, the level of these gasses rises at a rate comparable to that of CO_2, and can therefore be used to estimate the level of CO_2 (called the equivalent CO_2 or eCO_2).

The on-board ATtiny1614 microcontroller reads both sensors and controls the LED bar display and the buzzer. Via a serial protocol, a host system can request sensor readings or switch the LEDs and buzzer on and off. The kit's datasheet documents this simple protocol, so it will not be too difficult to write your own software to support the Air Quality Kit. As name of this kit indicates, it is designed for Raspberry Pi, but there is no reason why you shouldn't use it with other boards or systems with a 3.3 V UART.

The firmware of the ATtiny also offers an automatic mode (switched on by default) that shows the eCO_2 level on the LED bar without any external control; all that is needed is a 3.3 V power supply. So, even without a host system, the Air Quality kit can be used as eCO_2 monitor.

Software

As mentioned earlier, MonkMakes offers downloads of some Python example programs to control this Air Quality Kit to test and demonstrate all its features. In the *Getting Started* section of the documentation, the instructions clearly show how the software can be used on a Raspberry Pi board to implement an eCO_2-meter, an eCO_2-meter with acoustic alarm (**Figure 4**) and a data logger application. Looking at the examples, like in **Figure 5**, you'll notice that the ATtiny and the API completely relieve you of retrieving and evaluating sensor data: one simple instruction from the host (Raspberry Pi) will trigger the Air Quality board to echo the current ambient temperature (in °C) or CO_2 level (in ppm), respectively. Similar commands are used to switch the buzzer on and off, and to control the LEDs of the eCO_2 bar display.

A Nice Design

Only some basic knowledge of the Raspberry Pi is required to get this Air Quality kit working. What is a great advantage for some, will be less attractive for others: knowledge of the sensors and control of buzzer and LEDs is not required. The (source code of the) firmware of the on-board

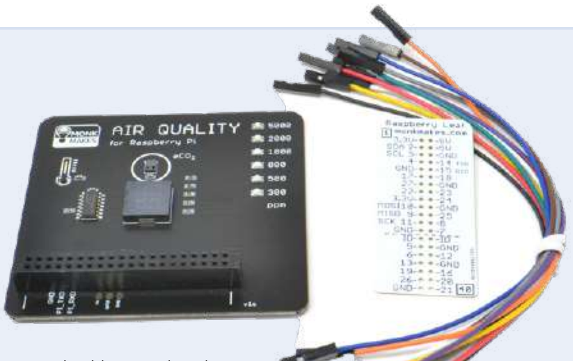


Figure 1: The kit contains the board, jumper wires and a template to locate the GPIO-connector pins. (Source: MonkMakes)

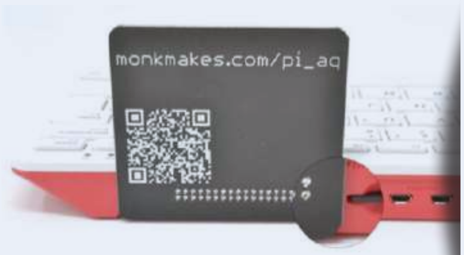


Figure 2: Air Quality Kit connected to a Raspberry Pi 400. (Source: MonkMakes)

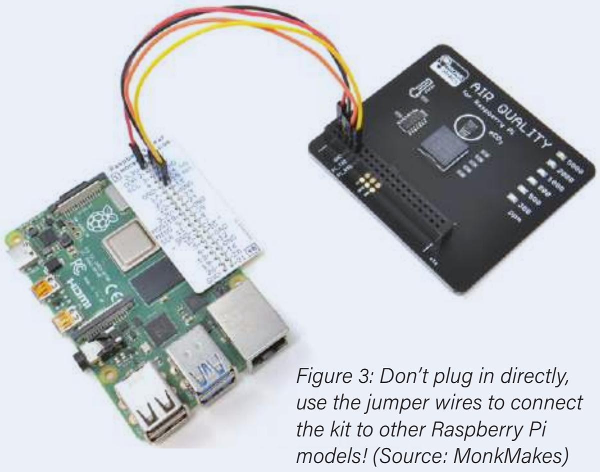
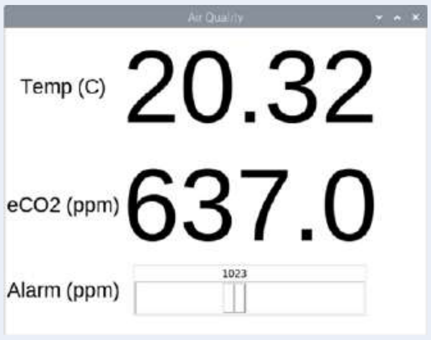


Figure 3: Don't plug in directly, use the jumper wires to connect the kit to other Raspberry Pi models! (Source: MonkMakes)

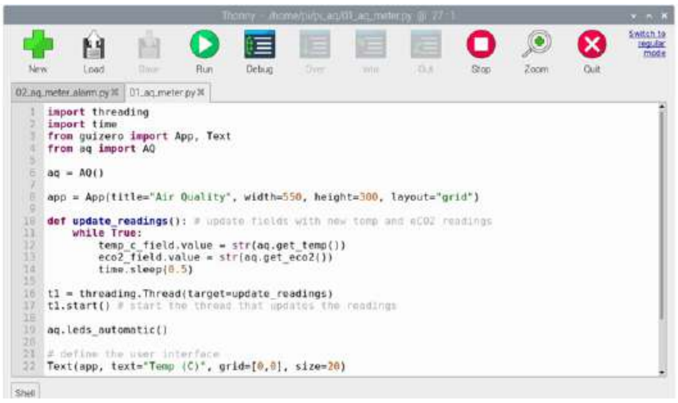


Temp (C) 20.32

eCO2 (ppm) 637.0

Alarm (ppm) 1023

Figure 4: Raspberry Pi screen output for one of the examples. (Source: MonkMakes)



```python
import threading
import time
from guizero import App, Text
from aq import AQ

aq = AQ()

app = App(title="Air Quality", width=550, height=300, layout="grid")

def update_readings(): # update fields with new temp and eCO2 readings
    while True:
        temp_c_field.value = str(aq.get_temp())
        eco2_field.value = str(aq.get_eco2())
        time.sleep(0.5)

t1 = threading.Thread(target=update_readings)
t1.start() # start the thread that updates the readings

aq.leds_automatic()

# define the user interface
Text(app, text="Temp (C)", grid=[0,0], size=20)
```

Figure 5: Python source, showing that only simple, short instructions are needed to communicate with the Air Quality Kit.

ATtiny1614 is not released (i.e., we don't know what exactly happens inside this microcontroller). However, the protocol to communicate with the board is simple and well documented, and development of your own applications — even for other target systems than Raspberry Pi boards — will be relatively easy. The MonkMakes Air Quality Kit for Raspberry Pi is a nicely designed, well-documented board that, together with the examples, is also suitable for beginners who want to get started with temperature and eCO_2 measurements. ◀

210681-01

Questions or Comments?

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

Contributors

Text: **Luc Lemmens**
Illustrations: **MonkMakes, Luc Lemmens**
Editors: **Jens Nickel, C. J. Abate**
Layout: **Harmen Heida**

Sensor Characteristics

eCO_2 minimum reading	400	ppm
eCO_2 maximum reading	4095	ppm
eCO_2 resolution	1	ppm
eCO_2 accuracy	unspecified	
Temperature minimum reading	-10	°C
Temperature max reading	100	°C
Temperature accuracy	+/- 2	°C

About CO_2 Concentrations

The level of CO_2 in the air we breathe has a direct influence on our well-being. CO_2 levels are of particular interest from a public health point of view. To put it simply, they are a measure of how much we are breathing other people's air. We humans breathe out CO_2, and if several people are in a poorly ventilated room, the level of CO_2 will gradually increase — as well as concentration of aerosols that spread colds, flues and Coronavirus. Another important impact of CO_2 levels is in cognitive function — how well you can concentrate and think.

The table below shows the levels at which CO_2 can become unhealthy. The CO_2 readings are in ppm (parts per million).

Level of CO_2

250-400	Normal concentration in ambient air.
400-1000	Concentrations typical of occupied indoor spaces with good air exchange.
1000-2000	Complaints of drowsiness and poor air.
2000-5000	Headaches, sleepiness and stagnant, stale, stuffy air. Poor concentration, loss of attention, increased heart rate and slight nausea may also be present.
5000	Workplace exposure limit in most countries.
>40000	Exposure may lead to serious oxygen deprivation resulting in permanent brain damage, coma, even death

WEB LINKS

[1] MonkMakes webpage with instructions: http://monkmakes.com/pi_aq
[2] Software on Github: https://github.com/monkmakes/pi_aq
[3] TMP235 datasheet: https://www.ti.com/product/TMP235
[4] CCS811 datasheet: https://www.sciosense.com/products/environmental-sensors/ccs811-gas-sensor-solution/

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

Light Switch DeLux

A Solution for High-Precision Light-Controlled Switching

By Clemens Valens (Elektor)

Light-controlled switches are plentiful and retail for €10 or so, but most of them are changing their state somewhere in the twilight zone. Sometimes applications require better precision and more control than these cheap switches allow for. Do you need a lux-accurate light control? If so, this project is for you.

There exist many, many designs for light-controlled switches and most of them work great in the application they were designed for. These applications usually consist of switching on a light when the ambient light level drops below a certain threshold and switch the light off again when the light level increases. Sometimes a timer is added too.

You might think that every possible application is covered by these designs and yet this is not true. The reason is that they all lack precision. Based on an LDR or phototransistor, they tend to switch somewhere in the twilight zone. However, light levels vary much more than that.

Brightness Is Subjective

To humans, daylight intensity or brightness is fairly constant. Of course, we notice variations due to clouds and the sun, but we are not very sensitive to them. The reason for this is the eye's logarithmic response to brightness. On a cloudy day brightness can vary between 5,000 and 10,000 lux, yet it looks almost the same to us. Sunlight may result in levels of over 25,000 lux, which we notice, obviously, but we don't experience it as three or more times as bright.

Plants, on the other hand, are way more sensitive to light intensity than humans. Farmers know this, and they shine artificial light on some of their crop even during the day to improve its yield. On sunny days this is usually not needed, but on cloudy days it may help. To do this in an economical way, they therefore need light-controlled switches that can detect brightness differences with more precision than an LDR or phototransistor can do.

Today light sensors exist that convert brightness directly to a value in lux with resolutions of up to 16 bits. Some of these sensors not only measure lux, but also UV and white light intensity. With such a sensor, it is quite easy to build a high-precision light-controlled switch.

High-Accuracy Ambient Light Sensor

A popular light sensor is the VEML7700 from Vishay. This is a high-accuracy ambient light sensor with I²C interface and it can be found mounted on a small module for a few euros. From the same family we might also cite the VEML6075, a UVA and UVB light sensor also with I²C interface.

Because of its digital I²C interface the sensor does not need an analog-to-digital converter and can instead be connected directly to most microcontrollers. Its output data is available in two 16-bit registers: ambient light (also called 'ALS') and white light. White light covers a wide spectrum from 250 nm up to 950 nm. The ALS spectrum is much narrower, from about 450 nm to 650 nm as it is optimized for human perception. Plants are not human, so the output to use will depend on the application.

Sensor sensitivity is important too. The VEML7700 has a resolution of 0.005 lx per LSB and a maximum detection level of 167,000 lx (minimum is 0.01 lx). Such a wide range would require 25-bit values, but the device is only 16 bits; therefore, a sensitivity value can be specified (sometimes called 'gain') to bring things into range. Its high sensitivity allows the sensor to be used behind low transmittance (i.e., dark) surfaces and still obtain usable results.

For low-light conditions, the sensor features an integration setting of up to 800 ms. Finally, low and high thresholds can be set that can trigger interrupts, making it easy to create alarms or allow for automatic switching.

Stick It on an ESP32

The VEML7700 module used for my experiments was bought at Adafruit. A good platform to use it with is the ESP32-Connected Thermostat [1] (a.k.a. Automator, see [2]). The module has five pins, but as it has two power supply options, we only need four of them. These four pins have the same order as the OLED display connector on the thermostat, and so we can stick the module on K9. Make sure the VIN pin remains unconnected and the module points upwards (as opposed to how the OLED display would be plugged, see **Figure 1**).

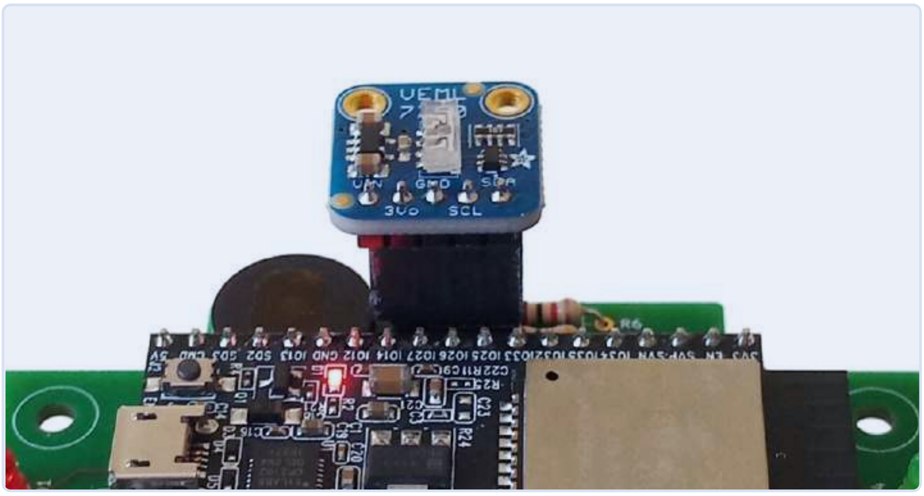


Figure 1: The 5-pin VEML7700 module plugs onto 4-way connector K9. Its VIN pin is not connected even though it may look like it.

Software With ESPHome

After connecting the sensor to the ESP32, we must produce some software to make it all work. As the objective is some sort of light-controlled switch, a logical choice would be to use a home automation platform and my favorite is ESPHome. Elektor has published several projects that used ESPHome [3][4], and so you may already know how to use and configure it, but this project introduces a concept that I didn't treat before: creating a custom sensor. If you are new to ESPHome, I recommend reading and watching [3] and [4] first.

We Need a Custom Sensor

ESPHome supports many sensors, but (at the time of writing) not the VEML7700. It knows of other lux sensors, but not this one. However, this is not a problem as ESPHome offers a method of adding your own sensor. Doing so involves writing some C++ code, so it makes things a bit more involved.

As before (see [3] and [4]) we must declare a *sensor* section in the ESPHome configuration file for this device. The sensor's *platform* now must be *custom*. This tells ESPHome that you will provide all the details for it.

Next follows a *lambda* section which consists of a few lines of C++ code to tell ESPHome to register a sensor of our own devising (that we named *veml7700*). We must also specify the data stream(s) that our sensor produces. In this case it has three outputs: ALS, lux and white light. The order is important and must be respected whenever they are referenced elsewhere in the YAML file.

We continue with normal YAML statements to further specify the sensor outputs. This is done with a *sensors* section (plural!) where we can specify for each data stream its name, its units and how many decimals to use. The order is the same as in the return statement in the lambda section.

Only the lux stream has units ('lx'), and there is no point in having decimals for any of them, so we set them to zero.

The last thing to be done is to indicate where ESPHome can find the driver for the custom sensor. We do this in the *esphome* section at the top of the configuration file where we add a file (veml7700.h) to the *includes* subsection. On the system that runs ESPHome, this file should be in the same folder as the device's YAML file.

The Hard Part

Done? Not really. Now comes the hard part, which is, of course, creating the driver for the custom sensor. This driver must comply to ESPHome standards for components (a sensor is a component), meaning that it must provide certain functions that ESPHome expects from its components, and it must communicate with the sensor itself.

For the second part, we can rely on Adafruit who, besides manufacturing the VEML7700 module, also created an Arduino library for it [5]. Our driver must provide the interface between ESPHome and the Arduino library.

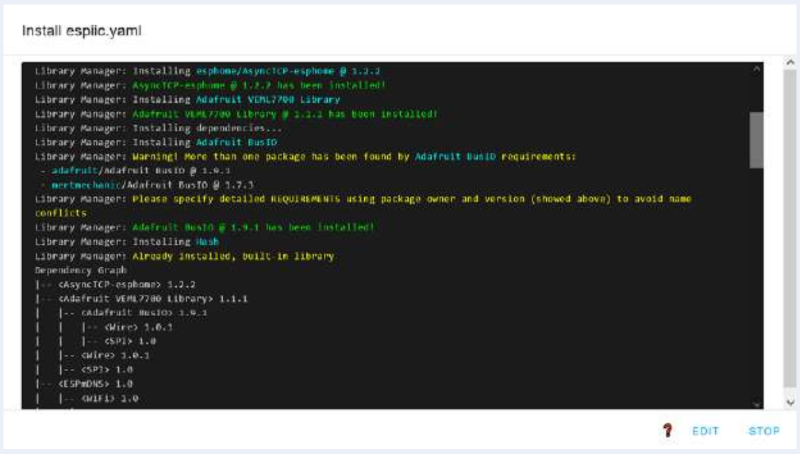
Installing a Third-Party Library

ESPHome provides a mechanism for adding open-source community libraries: simply add the exact (!) name of the library to the *libraries* subsection in the *esphome* section. In our case this is "Adafruit VEML7700 Library". Now, when ESPHome processes the configuration file, it will first install the library (if it didn't do so already, see **Figure 2**) before compiling everything. In your custom driver, you can simply include the library as any other library.

I haven't pushed this feature too far, so I don't know the criteria a third-party library must fulfill to be used this way. You may want to consult the platform.io documentation for more details as it is the toolchain used by ESPHome.

An Encapsulated Arduino Sketch

Our driver is a C++ class that must provide at least a constructor and a function named *update*. We need a function *setup* too so that we can initialize the Adafruit driver. The functions *setup* and *update* of our


Install espiic.yaml

```
Library Manager: Installing esphome/AsyncTCP-esphome @ 1.2.2
Library Manager: AsyncTCP-esphome @ 1.2.2 has been installed!
Library Manager: Installing Adafruit VEML7700 Library
Library Manager: Adafruit VEML7700 Library @ 1.1.1 has been installed!
Library Manager: Installing dependencies...
Library Manager: Installing Adafruit BusIO
Library Manager: Warning! More than one package has been found by Adafruit BusIO requirements:
 - adafruit/Adafruit BusIO @ 1.9.1
 - mertmechanic/Adafruit BusIO @ 1.7.3
Library Manager: Please specify detailed REQUIREMENTS using package owner and version (showed above) to avoid name
conflicts
Library Manager: Adafruit BusIO @ 1.9.1 has been installed!
Library Manager: Installing Hash
Library Manager: Already installed, built-in library
Dependency Graph
|-- <AsyncTCP-esphome> 1.2.2
|-- <Adafruit VEML7700 Library> 1.1.1
|   |-- <Adafruit BusIO> 1.9.1
|   |   |-- <Wire> 1.0.1
|   |   |-- <SPI> 1.0
|   |-- <Wire> 1.0.1
|   |-- <SPI> 1.0
|-- <ESPmDNS> 1.0
|   |-- <WiFi> 1.0
```

Figure 2: ESPHome installs the Adafruit library automatically.

class do what would normally be done in the functions *setup* and *loop* of a typical Arduino sketch using the Adafruit driver. Basically, our class encapsulates an Arduino sketch including global variables and adds some ESPHome-specific things to it. For ESPHome we must insert calls to *publish_state* for each data stream (ALS, lux & white light) to the function *update*. This will make the data available to the rest of the world. The call order must be the same as in the return statement in the *lambda* subsection of the *sensor* section (see above).

As our class inherits from the *PollingComponent* class which itself inherits from the *Component* class, other functions are available that you might want to use. A polling component is a component that is called periodically, and the call rate can be specified (e.g., in our constructor). Refer to [6] for more details.

Adding the I²C Bus

The one thing that remains to be done now is connecting the sensor to the I²C bus inside the software (i.e., create a logical connection between the two, as we already have a physical connection). The Adafruit library uses Arduino's Wire library for this purpose. In the ESPHome YAML file, we therefore add an *i2c* section so ESPHome knows that it is needed.

The ESP32 has two I²C buses with SDA and SCL signals that may be connected to almost any pin on the chip. ESPHome therefore supports multiple I²C buses with freely assignable pins, making it a matter of specifying what goes where. But the Arduino Wire library supports only one I²C bus, so how do you tell it to use the bus you want? Well, you can't, as it will always use the default bus. In ESPHome the default I²C bus is the first bus specified in the *i2c* section. It would have been nice to be able to specify an I²C bus by using its ID, and ESPHome would probably be more than happy to let you do it, but the underlying Arduino Wire library doesn't allow this.

Done

Here ends this article. If the Automator is programmed by ESPHome with the code described above, you will get a device that can be controlled from a home automation controller like Home Assistant (or by itself). There are no automation rules inside the ESPHome YAML file, so without a controller it will just measure ambient light intensities. Automation rules can be added to the YAML file itself, or they can be created inside e.g. Home Assistant. For a description of the rest of the YAML code, which is nothing special, please refer to [3] and [4].

The YAML configuration file and C++ code is available for download from this article's the webpage [7].

210190-01

Contributors

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

Questions or Comments?

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

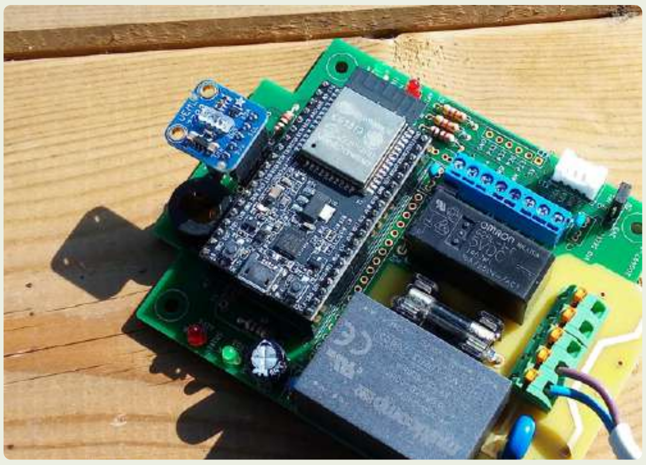
RELATED PRODUCTS

- ESP32 DevKitC (SKU 18701)
  www.elektor.com/18701
- 0.96" 128×64 I²C OLED display (SKU 18747)
  www.elektor.com/18747
- Koen Vervloesem, *Getting Started with ESPHome* (SKU 19738)
  www.elektor.com/19738

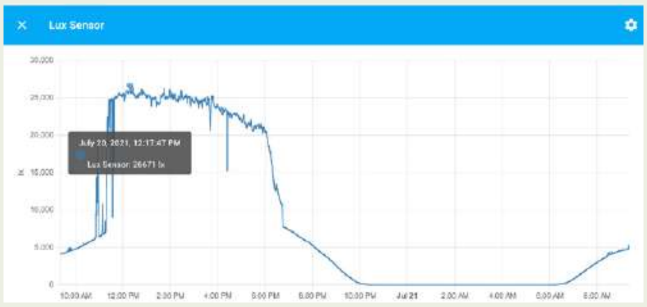
WEB LINKS

[1] Y. Bourdon, "ESP32-Connected Thermostat," ElektorMag 9-10/2021: www.elektormagazine.com/200497-01
[2] Elektor Automator: www.elektormagazine.com/labs/automator
[3] ESPHome starts here: www.elektormagazine.com/labs/how-to-home-assistant-esphome
[4] C. Valens, "Home Automation Made Easy," ElektorMag 9-10/2020: www.elektormagazine.com/200019-01
[5] Adafruit VEML7700 library: https://github.com/adafruit/Adafruit_VEML7700
[6] ESPHome on GitHub: https://github.com/esphome
[7] Downloads for this article: www.elektormagazine.com/210190-01

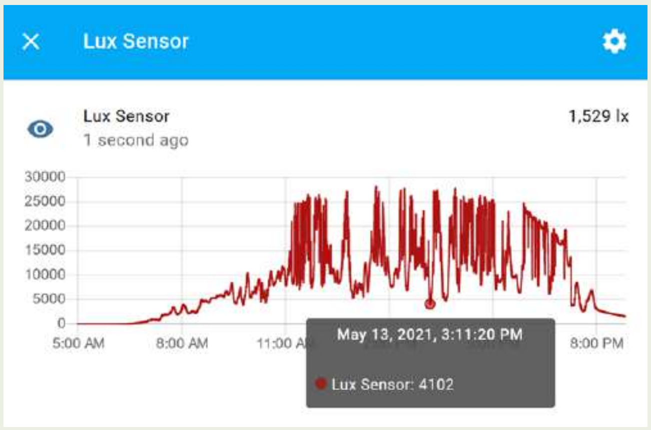
Some Results



The light sensor placed in bright summer sunlight. At noon, with sensitivity set to 0.125 and an integration time of 100 ms, the sensor reported values of around 48,000 for ALS and a bit more than 30,000 for white light. These values correspond to light intensities of around 22,000 lx. At 10:30 AM with the same conditions, I measured 16,800 lx, a difference of 5,200 lx. However, to my eyes the brightness looked pretty much the same in both situations.



Evolution of brightness on a sunny day in July. The “bouncing” on the left is due to the shadow of a tree. From noon to 6 PM the brightness slowly decreases but you wouldn’t really notice it. The dips are caused by clouds. The shadow of the house starts to cover the sensor at 6 PM, which explains the steep fall. Then light intensity continous to decrease until dark. For humans 5,000 lx is already bright.



On a partly cloudy day in May brightness levels may be anywhere between approximately 4,000 and 27,000 lx.

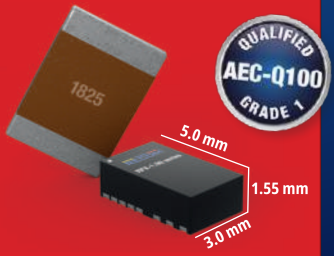
All graphs were created with the help of Home Assistant at a latitude of 47°N.

Advertisement

AUTOMOTIVE-GRADE POWER MODULES

Smaller than an 1825 capacitor

- Wide operating temperature from -40°C up to +125°C AEC-Q100 Grade 1
- 36VDC Vin / 0.5 or 1.5A Iout
- Excellent performance for automotive systems
- Optional wettable flanks for easy optical inspection
- Adjustable output voltage
- High efficiency, no need for heatsinks
- High power density in compact QFN packaging
- Integrated FETs, inductors, and passives for simple design
- Fully-protected with SCP, OCP, OTP, and UVLO
- Simple class A or B filtering



RECOM

WE POWER YOUR PRODUCTS

recom-power.com/mobility

The Challenges in Bringing **IoT Solutions** to Market

Worries Around Security, Scalability, and Competition

Figure 1: The European Commission has examined whether the big players in voice assistants, the favorite user interface to the smart home, are stifling competition. (Source: ShutterStock/Gorodenkoff)

By Stuart Cording (Elektor)

The Internet of Things (IoT) has been around for more than 20 years, and a host of wireless technologies have sprung up to support its deployment. In the home, voice assistants have established themselves as the primary user interface to an array of smart devices. Despite these advancements, more than a third of IoT projects never make it past the proof-of-concept phase. Furthermore, the European Commission is concerned that a lack of competition in some application spaces may be hindering market entry for EU businesses. So, what are the real challenges, and what is it like to deploy an IoT solution across Europe?

If you want to get a feel for the technologies behind the Internet of Things (IoT), finding a raft of suitable projects doesn't take long. Just search for "IoT" and you're preferred prototyping platform, and you'll be overrun with pages of projects, cloud platforms, technology comparisons, and lists of ideas. *Elektor* is also a great source of information. Since the term IoT was coined more than 20 years ago, our website has collected more than 600 articles on or related to the topic [1].

However, there is a considerable gap between a prototyped platform that demonstrates the core concept of an IoT solution and deploying one for real. The IoT Insights report from Microsoft [2] interviewed more than 3,000 IoT professionals in 2021. They found that 35% of IoT projects experience failure during the trial or proof-of-concept phase, up 5% on their survey from a year earlier. Most cited as the reason for failure at this stage is the high cost of scaling. Other reasons include too many platforms to test, too many use cases to prove, and a lack of resources. A separate study by Cisco [3] reported that only 26% of companies surveyed thought their IoT initiatives had been successful. Responses to their study showed that, while most IoT projects look good on paper, they proved to be more complex than initially thought.

Despite such gloomy feedback on IoT overall, there are sectors where IoT is making huge inroads and delivering growing revenues.

EU Commission Reviews Consumer IoT Sector

EU citizens have welcomed the range of consumer IoT solutions on offer in recent years. So much so, a Smart Home revenue report from Statista [4] predicts that related revenues will double from around €17b in 2020 to around €38.1b in 2025. Concerned that competition in this sector may be being stifled, the European Commission undertook an inquiry as part of their digital strategy [5]. The report, released in January of 2022, garnered input from manufacturers of wearable devices, connected consumer devices used in the smart home, and those providing services over such smart devices. Additionally, the Commission requested input from standard-setting organizations. However, upon reading, many paragraphs of their analysis are given to the voice assistants (VA) that form the user interface to many IoT products and services (**Figure 1**).

Looking at the available IoT landscape, it is clear that business opportunities abound, regardless of whether you are focused on solutions for consumers or industry.

Having analyzed the landscape of the smart home, the report [6] makes it clear that, in Europe, Google's Google Assistant, Amazon's Alexa, and Apple's Siri are the leading general-purpose VAs. Other VAs are available, but these tend to be more limited in functionality and focus on supporting a single product or a service provider's app. According to ZDNet, of the solutions from the big three players, Amazon provides the highest level of compatibility, supporting around 7,400 brands [7]. By comparison, Google supports around 1,000, while Apple remains the most exclusive, supporting about 50.

Little Room for Voice Assistant Newcomers

With such powerful industry players already established on the market, there is little room for newcomers, and the technology curve to develop a competitive VA is significant. Thus, if you want to leverage voice control for your IoT solution, you have to play according to the rules set by the big three. An alternative approach would be to license a VA. However, some manufacturers questioned reported that licensing conditions were restricting their choices. These ranged from exclusivity or restrictions to stop more than one VA being used concurrently to licensing that forced the inclusion of other types of software or applications, meaning the VA technology couldn't be used standalone.

Another big concern is access to data. As a third party relying on a VA, you only have limited access to the data collected. The VA provider has access to the audio recordings and would also know how many failed attempts there have been to issue the commands selected for your device. However, your team will not be provided access to the audio recordings, so you'll more likely have to wait for user feedback to discover that your choice of voice commands is suboptimal amongst a group more expansive than that used for testing. Additionally, because the VA provider can analyze everything spoken into it, they could conceivably use this data to develop a solution that competes with yours or leverage user experience provided by your IoT solution to improve their services.

A further issue arises when the VA provider also offers advertising services. Theoretically, the voice input provided by your users could help the provider target advertising more accurately to the demographic represented by your customer base.

Finally, there is the loss of brand recognition and experience. Your carefully crafted solution is at the whims and mercies of the VA. Should they choose to make a significant change, such as the voice used, the wake word, or even roll-out functionality changes that result in a drop in users, you're inevitably going to suffer the consequences too.

The report also examines many other relevant areas, including application programming interfaces (API), standards, interoperability, the imbalance in power between many third-party IoT device developers and the big cloud platform service providers, and contract termination clauses.

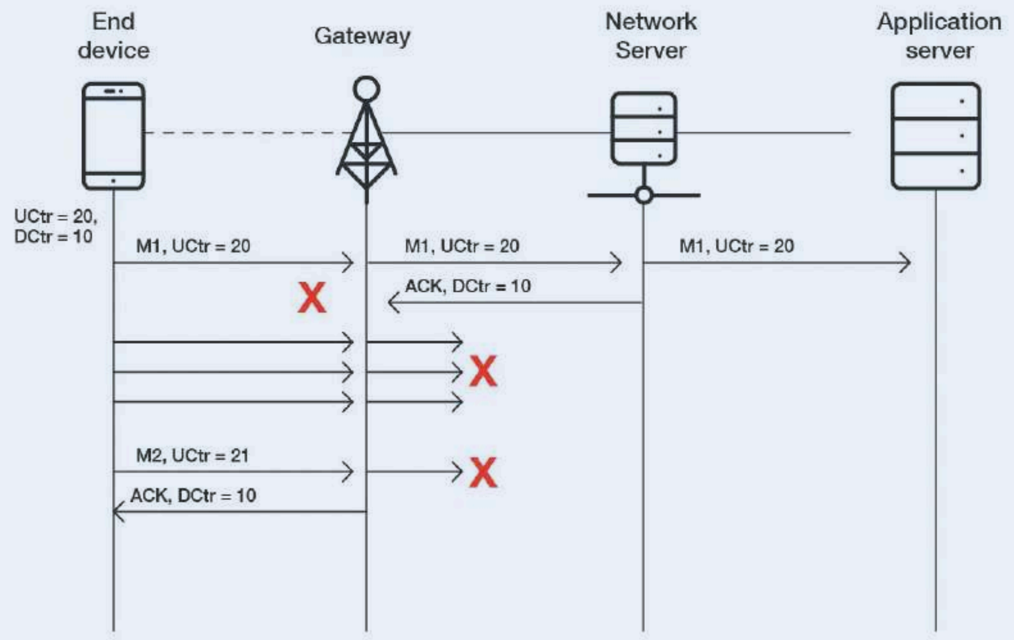


Figure 2: Researchers discovered a Denial-of-Service (DoS) attack in LoRa 1.0. By replaying a previous successful data transfer, a LoRaWAN node is hindered from sending further data packets. (Source: Trend Micro)

There are no recommendations in the report. However, the conclusions state that the report's content will contribute to the Commission's standardization strategy and feed into the legislative debate on the Digital Markets Act (DMA).

Security Concerns Worry IoT Implementers

Reviewing the data collected in Microsoft's IoT Insights report, it is IoT security that is high on the list of worries. This issue is of particular concern for those planning IoT solutions, with 29% indicating that the associated security risks are holding them back from using IoT more. The report also explains that around a third of organizations are concerned about the security risks of IoT, especially data breaches. To combat this, outsourcing is highlighted as the best way of improving peace of mind.

While many engineers know the phrase "obscurity is not security," few are skilled enough in this domain to ensure a solution is protected, end-to-end, from attackers. And while semiconductor suppliers offer a range of single-chip security solutions, developers still need to understand how to use them correctly to ensure that they don't inadvertently introduce new security weaknesses.

Over the past decades, low-power wide-area networks (LPWAN) solutions such as LoRa and Sigfox have established themselves as key wireless IoT technologies supporting long-range communication. Achieving ranges of tens of kilometers, they are an alternative to cellular wireless such as LTE Cat-M1 and NB-IoT thanks to their superior low-power performance for low data volumes [8]. But how secure are they?

LoRaWAN Under the Magnifying Glass

LoRa and LoRaWAN have been subject to particular scrutiny by the security and hacking community. Sébastien Dudek of Trend Micro, a company focused on IT security solutions, is one of several researchers that has written extensively on some of the potential issues. In a series of three technical briefs [9][10][11], he outlines a range of issues in the implementation and potential attacks. These range from denial of service (DoS) (**Figure 2**) and eavesdropping to bit-flipping (**Figure 3**) and spoofing of acknowledgments (**Figure 4**). The outcomes of such attacks range from the inability to communicate with nodes and reducing the battery life to altering application data.

Many of the vulnerabilities highlighted were resolved between version 1.0.2 and 1.1 of the LoRaWAN standard. However, further challenges arise when operating LoRaWAN nodes with gateways using different versions of the specification. In such cases, there is a need to make modifications to ensure secured backward compatibility between end devices and the back-end, as highlighted in a paper from 2018 by Tahsin C. M. Dönmez [12].

Beyond hacking the wireless link, there is also the issue of bad actors stealing and directly attacking the hardware. Sébastien Dudek also examines this aspect of security. In the case of LoRaWAN, many solutions use a microcontroller (MCU) and a Semtech wireless module. As these are connected via SPI, data passing between the two can be easily captured and analyzed.

Beyond this, there is also the issue of the security of the MCU itself. One attack method simply extracts the firmware from flash

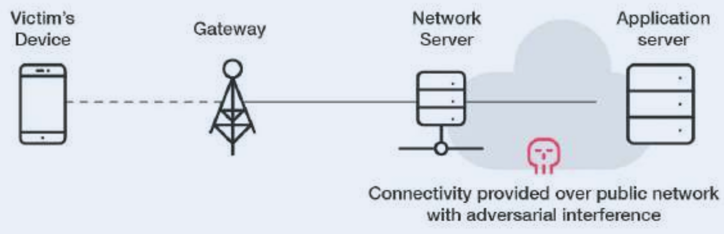

Victim's Device
Gateway
Network Server
Application server
Connectivity provided over public network with adversarial interference

Figure 3: Due to the known fixed structure of LoRaWAN data packets, it is possible to flip bits (bit-flipping attack) in the content without having to decrypt the message. This requires access to the network server receiving the data. (Source: Trend Micro)

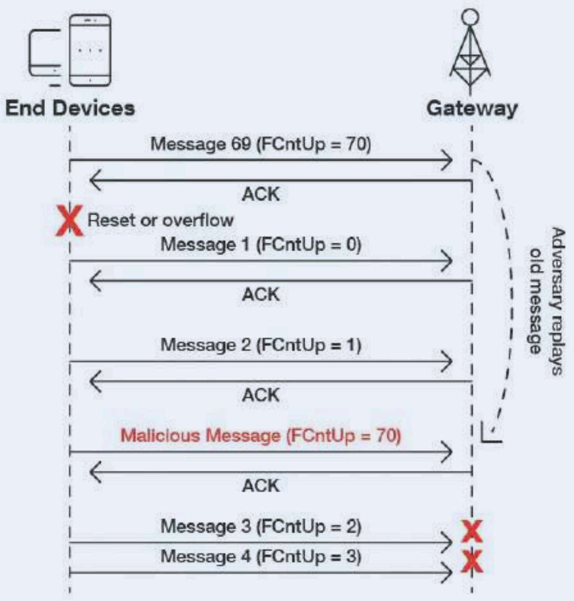


Figure 4: Jamming the wireless link results in the LoRaWAN node repeating the packet transfer up to seven times, reducing battery life. If the attacker also captures and replays acknowledgment packets (ACK), the node thinks the link is still functional as ACK packets don't declare the message they are acknowledging. (Source: Trend Micro)

memory, allowing the code to be analyzed. If the security keys are also in the firmware, an attacker can use them to develop nodes that spoof authentic end devices. The recommendation to counter this is the use of Secure Elements (SE), single-chip authentication devices that securely store encryption keys. An approach using Microchip's ATECC608A [13] is one of several for which example code is available. However, while example projects demonstrate how to protect the cryptographic keys, the secure boot feature of this authentication device is not used. Thus, if the same approach were used for a product, the authentication device could be removed and used as an SE with a different MCU and new firmware.

Security Issues Across the Board

LoRaWAN end-devices provide only limited data bandwidth and, as they have no IP address like a Wi-Fi module, they are not addressable. As such, they offer minimal risk to corporate networks. However, potential risks lie with applications based upon such wireless technologies. These can have implications on lives and the environment should something goes wrong. For example, as part of a smart city network, LoRa-based sensors may be responsible for monitoring water levels to avoid flooding. If the data from the sensors are blocked, flood defense systems may not respond. Conversely, false injected data could result in flood defenses responding to an event that isn't happening with potentially equally disastrous consequences.

And while LoRa and LoRaWAN have been highlighted here, plenty of researchers are examining other LPWAN technologies, including Sigfox and NB-IoT. In a paper by Florian Laurentiu Coman et

Figure 5: DEUS POLLUTRACK particulate matter sensors gather data from stationary and mobile IoT units. Data is delivered to the back-end using 4G and 5G cellular networks. (Source: DEUS POLLUTRACK)

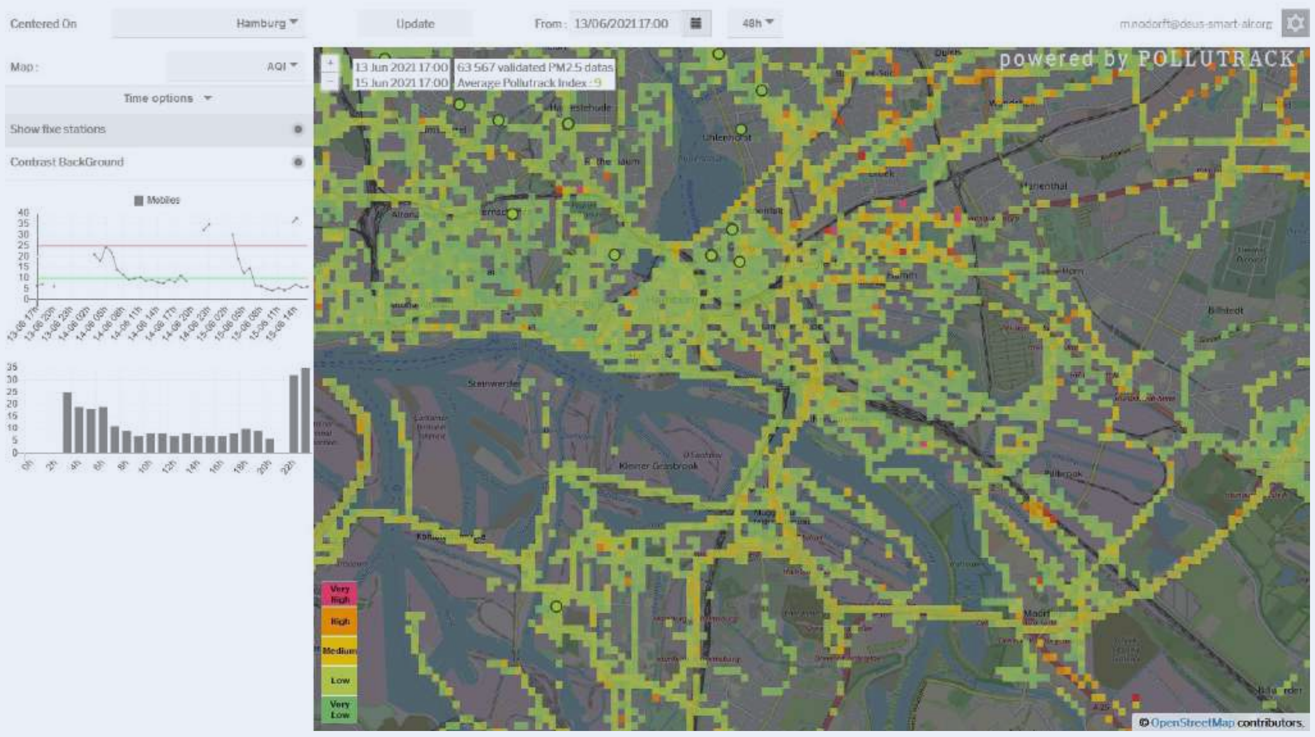


Figure 6: A cloud-based dashboard shows the level of airborne pollutants, as shown here for the German city of Hamburg. Local authorities use such data to make decisions on transportation solutions. (Source: DEUS POLLUTRACK)

al. [14], several proof-of-concept attacks on wireless networks in addition to LoRaWAN are described. In a technical brief issued by Deutsch Telekom [15] it was stated that implementing Sigfox and LoRaWAN "without [an SE] can [even make] end-to-end encryption useless." It explains that, by contrast, NB-IoT benefits from long-proven LTE security features, such as authentication and secure key generation and exchange. However, it also makes clear that end-to-end encryption is not standard and, if deemed necessary, must be discussed with the network operator.

Delivering City-Wide IoT Solutions

Concerns regarding security in LPWAN networks influenced technology choices made by DEUS POLLUTRACK Smart City GmbH i.G. for their IoT platform [16]. Their team has been developing IoT sensor networks to monitor particulate matter in cities for more than a decade. With the technology deployed in more than 15 European cities, it enables local leaders of municipalities to make informed environmental decisions regarding air pollution. Their patented optical particle counters (OPC) are capable of monitoring down to the ultrafine particle (UFP) classification (under 0.1 µm). While larger particulates, such as PM_{10} are considered dangerous for the lungs, UFPs can enter the bloodstream and pass to other organs through inhaled air.

DEUS's sensor technology (**Figure 5**) uses a combination of stationary and mobile sensors, networked to back-end dashboards that visualize the data collected. Cities such as Marseille and Paris use 40 stationary sensors complemented by 300 mobile sensors [17]. Mobile sensors are fitted to vehicles of partners, such as the delivery vans of DPD, that are already frequently underway in the target city. These sensors recalibrate themselves against data acquired by the stationary sensors they pass to ensure the accuracy required based. All this requires a robust, reliable, and secure LPWAN choice.

Talking to co-founder Marc Nodorft, he explains that both Sigfox and LoRaWAN were considered during the early development stages. Sigfox offered connectivity infrastructure, simplifying system deployment, but neither provided the data throughput required. LoRaWAN, at the time, wasn't secure enough out-of-the-box and, without infrastructure partners in the cities where the technology was to be deployed, it would be necessary to deploy gateways that connected to the back-end via cellular networks. As a result, 4G and, later, 5G cellular were selected, resolving the issues of coverage, reliability, and security to the levels required.

Nodorft also tells us that, while there are plenty of cheap electronic solutions for IoT available, these are not robust enough for long-term deployment in the environments where their products are installed. Hence the choice was made to develop according to industrial standards, another consideration for those planning their own IoT products.

Another aspect is the back-end operations, which they have developed specifically to the needs of their IoT implementation (**Figure 6**). Moving forward, there is a need to support open-source reporting dashboards to allow government bodies using the system and citizens to access the data, which requires a cloud services provider. And, while there are plenty of choices, the provider is considered as important as the technical solution. Thus, the search is on for a provider that can provide personal support and not just an impersonal customer service chatbot.

With so much experience in significant IoT deployments and learning much about the technical challenges, I wondered what other advice Nodorft could offer those seeking to implement IoT solutions. "We've always stayed true to our vision," he replies, "which has often required us to change the approach." This has meant evaluating different technologies, working with different partners, and modifying the sales strategy on their road to success.

Teams of Experts and Partnerships Required

Looking at the available IoT landscape, it is clear that business opportunities abound, regardless of whether you are focused on solutions for consumers or industry. However, the journey from concept to deployment is fraught with challenges. While embedded systems developers may be well versed in hardware and firmware development, and may even have experience in wireless technologies, IoT and its security and scalability challenges may be too much for an organization to tackle alone.

According to the European Commission's report, large organizations also have the upper hand in business relationships when it comes to services and platforms. Small players and startups will struggle to get the support they need in these asymmetrical relationships if they go it alone. Without a doubt, expertise, either hired or loaned, is essential to move beyond example applications, demonstration dashboards, and test IoT services. Finally, it is vital to fix your vision while remaining agile in all areas of implementation, from technology choices to target market, to fulfill it.

220053-01

Contributors

Text: **Stuart Cording**
Editors: **Jens Nickel, C. J. Abate**
Layout: **Harmen Heida**

Questions or Comments?

Do you have technical questions or comments about this article? Email the author at stuart.cording@elektor.com or contact Elektor at editor@elektor.com.

WEB LINKS

[1] Elektor IoT Articles: www.elektormagazine.com/select/internet-of-things-IoT
[2] "IoT Insights, Edition 3," Microsoft/Hypothesis, October 2021: https://bit.ly/3rxMk3a
[3] "The Journey to IoT Value," Cisco, May 2017: https://bit.ly/3GzdJWS
[4] Dr. J. Lasquety-Reyes, "Smart Home - revenue forecast in Europe from 2017 to 2025," Statista, June 2021: https://bit.ly/3LlGiuG
[5] "Sector inquiry into the Consumer Internet of Things," European Commission, January 2022: https://bit.ly/3Lgw9iE
[6] "Final report - sector inquiry into consumer Internet of Things," European Commission, January 2022: https://bit.ly/3B2Htu9
[7] "The best voice assistant," ZDNet, September 2021: https://zd.net/3rxf6Rt
[8] L. Tan, "Comparison of LoRa and NBIoT in Terms of Power Consumption," KTH Royal Institute of Technology, January 2020: https://bit.ly/3JafsUb
[9] S. Dudek, "Low Powered and High Risk: Possible Attacks on LoRaWAN Devices," Trend Micro, January 2021: https://bit.ly/3rA02Tg
[10] S. Dudek, "Gauging LoRaWAN Communication Security with LoraPWN," Trend Micro, February 2021: https://bit.ly/3LhV0T5
[11] S. Dudek, "Protecting LoRaWAN Hardware from Attacks in the Wild," Trend Micro, March 2021: https://bit.ly/3rxquge
[12] T.C.M. Dönmez, "Security of LoRaWAN v1.1 in Backward Compatibility Scenarios," Elsevier, 2018: https://bit.ly/3GtzKq0
[13] Microchip Product Page - ATECC608A: https://bit.ly/3B7zlms
[14] F.L. Coman et al., "Security issues in internet of things: Vulnerability analysis of LoRaWAN, sigfox and NB-IoT," IEEE, June 2019: https://bit.ly/3uwhUQX
[15] "NB-IoT, LoRaWAN, Sigfox: An up-to-date comparison," Deutsche Telekom AG, April 2021: https://bit.ly/3uyUydj
[16] DEUS Pollutrack Website: https://bit.ly/3sHL9O5
[17] DEUS Sensor Measurement Network: https://bit.ly/3Gzjbcc

Miniaturisation of **Electronic Components and Industrial Sensors**

By Richard Woodward, Distrelec

Electronic components are getting smaller and smaller. Several inter-related factors explain this trend. The pressure is on end-product manufacturers to make their products smaller yet increase their capabilities and functionality. To achieve that, component manufacturers need to keep innovating their technologies to fit into a smaller space. Technology research and development makes this possible; an excellent example is how semiconductors continue to pack billions of transistors into an ever-decreasing footprint. But semiconductors are not the only electronic component that needs to get smaller. Today, most electronics-based end products may only contain less than five semiconductors, even though their capabilities have hugely increased. However, supporting and equally essential components, such as passives (inductors, capacitors, and resistors), are required in the hundreds.

Component miniaturisation requirements are not limited to just electronic components. Factory floor space is at a premium in the industrial domain, and the space available for each production asset, sensor and actuator are limited. Consequently, the next generation of each asset needs to be smaller and provide more functionality.

Component Miniaturisation Strives Forward

Change is constant in the electronics industry. Since the birth of the transistor in the late 1940s, researchers embarked on an evolutionary journey that would focus on miniaturisation. A decade later, the first integrated circuit (IC) containing four transistors was developed by Robert Noyce at Fairchild Semiconductor. Fast forward to today, and you'll find tens of billions of transistors in leading-edge processors. The process of fabricating semiconductors in an increasingly smaller area has advanced significantly. However, the advancements in component design and manufacture have also benefitted the broader electronics industry.

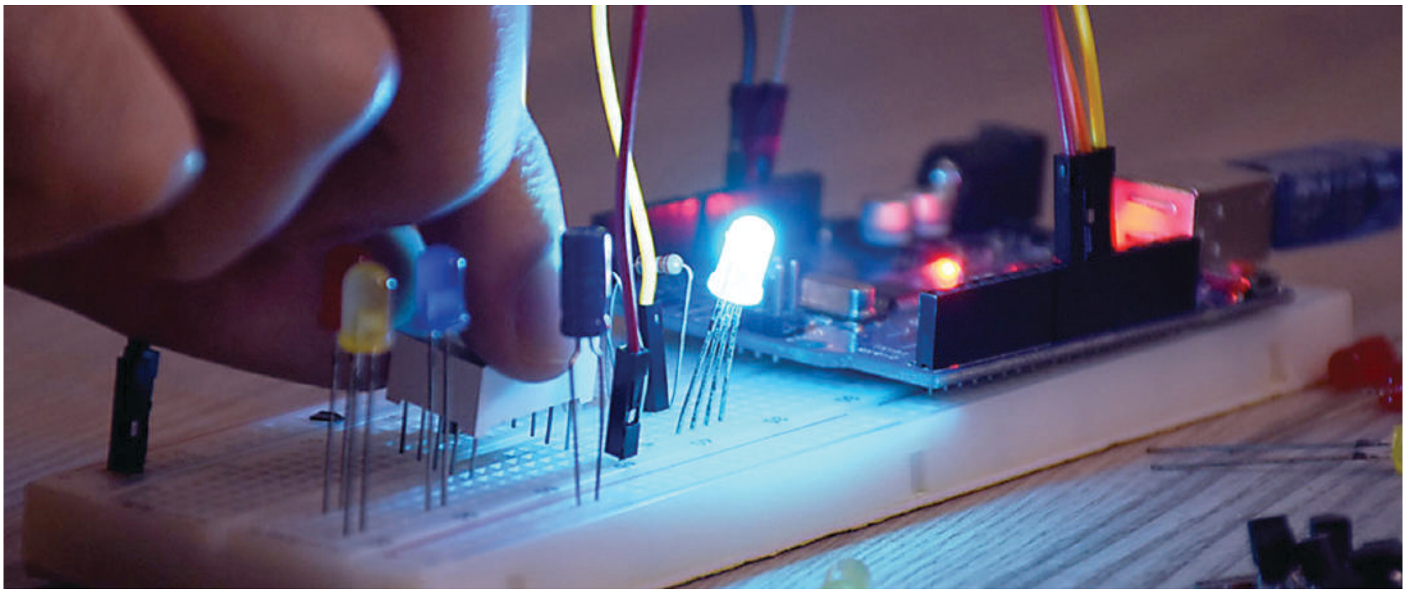
The Impact of Technology Advancements for Electronic Components

The first transistor prototype was a large item compared to today's devices. However, it was significantly smaller than the legacy technology of that era: thermionic valves. Not only was the transistor more diminutive, but its supply voltage arrangement was less complex than used on valves. Also, without the need for a heater element, the circuit ran cool. At an early stage, engineers could see the potential of what the integration of transistors into an IC could achieve. The journey to increase the capabilities of an IC while reducing the physical footprint had started. Gordon Moore, the co-founder of Intel, famously forecast his 'Moore's Law', that the 'number of transistors incorporated into an IC would double every two years'.

The research and development into semiconductor IC design and development continues to lead the electronics industry today. Making electronic components smaller challenges the automated production equipment manufacturers to accommodate smaller sizes. The physical dimensions now involved are staggering. For example, an advanced semiconductor process node size is currently 5 nm. This dimension doesn't relate to the actual transistor size but is used by semiconductor manufacturers to indicate the transistor density. Many smartphones use ICs based on the 5 nm process, and the computation power of its 30 billion transistors enables the phone's operation and all our popular apps.

Transistors and semiconductor ICs are packaged according to standard surface mount (SMT) packaging specifications managed by the JEDEC Solid State Technology Association. The same is true for surface-mounted passive components such as capacitors, resistors, and inductors. As semiconductors shrink and their capabilities increase, the need for their supporting passive components to reduce in size is paramount. Take a look at any embedded system design today, and you'll spot just a couple of complex ICs. However, there will be many hundreds of passive components placed around the ICs that are fundamental to their operation.

An example transistor package format is SOT23-3 (**S**mall **O**utline **T**ransistor). Typically used for small signal general-purpose transistors, it has three terminals and measures 3 mm x 1.75 mm x 1.3 mm. Some ICs also use the SOT-23 package, the '-x' indicating the number of pins used. For example, a SOT23-6 denotes an IC with six pins.



Integrated circuits have a wide variety of different package configurations, some leaded, others not. Either way, they all are surface mounted. Some sensors, such as micro-electromechanical systems (MEMS), are constructed in popular semiconductor packages.

Examples include:

- SSOP (Shrink Small Outline Package) — leaded — with a pin pitch of 0.635 mm
- TSSOP (Thin Shrink Small Outline Package) — leaded — this has a pin pitch of 0.65 mm
- QFN (Quad Flat Non-leaded) package — this is available in a variety of different electrode (pin) connections (from 14 to 100) and various pitch widths (0.5 mm – 1.65 mm)

The majority of surface-mounted "chip" passive components use the EIA codes to denote component size. Popular examples sizes include:

- 0805, measuring 2.0 mm x 1.30 mm (0.08 inches x 0.05 inches)
- 0603, measuring 1.5 mm x 0.80 mm (0.06 inches x 0.03 inches)
- 0402, measuring 1.0 mm x 0.50 mm (0.04 inches x 0.02 inches)

Component Miniaturisation in the Industrial Domain

End-product manufacturers are keen to take advantage of the continual size reduction of electronic components. The industrial sector, in particular, has seen the degree of automation used increase significantly over the last decade. Industrial efficiency improvement initiatives, such as Industry 4.0 and the Industrial Internet of Things (IIoT), are responsible for the increasing deployment of electronics-based sensors, control equipment, and AI machine learning systems. This increase in automation equipment is set against the backdrop of available factory space at a premium, driving demand for compact, energy-efficient, and leading-edge component technologies.

The drive to component miniaturisation is not limited to electronic components. Manufacturers of automation equipment and sub-assemblies have also been innovating. Advances in 3D printing, material technology research and a deeper understanding of finite element analysis in mechanical design contribute to weight, size, and cost reductions.

The Benefits of Component Miniaturisation

The ongoing trend of component miniaturisation enables designers and manufacturers to incorporate more advanced features into their end-products while also reducing their footprint. Whether designing an IIoT sensor or an industrial programmable logic controller, you can realise space savings within the product enclosure. Miniaturisation allows for the design of more stylish and space-efficient products. It also opens up the opportunity to increase the features and functionality of new products without increasing their dimensions.

For further information visit *www.distrelec.com*.

220286-01

About Distrelec

Headquartered in Manchester (UK) and Nänikon (CH), the Distrelec Group is a leading online distributor of electronic and technical components in the B2B sector with around 500 employees. In addition to its main sales markets of Switzerland and Sweden, the company has a strong market position in a total of 17 European countries. Its product portfolio is particularly characterised by a strong focus on MRO components as well as a range geared towards at B2B customers.

Preferably **Wired** After All

Tips for Developing a 1 Gbit/s Interface in an Industrial Environment

By Dr. Heinz Zenkner (Freelance Consultant at Würth Elektronik)

Wireless networks are becoming increasingly popular, especially in industrial applications. However, there is a strong case for robust cabling via Ethernet in many cases as the more reliable and secure option. This article demonstrates how to easily implement a 1 Gbit/s interface.

Industrial wireless sensor networks can be established using smart sensors and meters that use efficient modulation and coding techniques with good propagation characteristics and low bandwidths. However, the majority of the use cases explored are limited to low throughput applications. For these use cases the actual throughput is often no more than 1 Mbit/s.

There are no definitive boundaries in a wireless network. For example, even minor adjustments to the access point's antenna positioning can have a significant effect on the signal strength at the other stations. The signal is attenuated by walls, ceilings, and floors, and reflected by metallic objects. While a station may be able to receive the signal from an access point, the access point in turn may not be able to receive the station's signal. In addition, there is a possibility that the network could be accessed from the outside or that the wireless signal transmission could be interfered with.

As a result, wireless data transmission is intrinsically less reliable than transfers through a wired network. Thus, particularly in industrial settings, there may be instances where a wired Ethernet network is the only viable solution.

Wired Ethernet Network

Similar to wireless networks, wired networks work by exchanging Ethernet frames between endpoints. There are a few rules to follow when setting up a network to avoid problems. The most common cause of network problems is rule breaches. For example, in Ethernet, wire length must not be arbitrary. When cascading, i.e., connecting hubs in series, an arbitrary number of hubs is not permitted, and an unfavorable network configuration can also lead to errors or add unnecessary loads on the network. However, depending on the cable quality and the performance of the hardware, the expected data rates are often not achievable.

Currently, 100Base-TX (100 Mbit/s Fast Ethernet), Gigabit Ethernet (1 Gbit/s), 10 Gigabit Ethernet (10 Gbit/s) and 100 Gigabit Ethernet (100 Gbit/s) are available. For most applications, Gigabit Ethernet works well with regular Ethernet cables, specifically CAT5e and CAT6 cabling standards. These cable types adhere to the 1000BASE-T wiring standard, alternatively referred to as IEEE 802.3ab.

The 1 GB Ethernet interface conforms to the 802.3ab-1999 (CL40) standard and requires four wire pairs/channels for signal transmission. This results in a symbol rate of 125 Megabaud (MBd) and a bandwidth of 62.5 MHz per channel (2 bits per symbol). The signal voltage at 1000BASE-T (GB Ethernet) is typically 750 mV differential, for the limits 820 mV > V_{Signal} > 670 mV at a load of 100 Ω.

1 Gbit/s Ethernet Front-End

A typical front-end for Ethernet is equipped with an RJ45 port. These are intended for full-duplex transmissions, i.e. simultaneous transmission of send and receive data. This is possible because the connector contains two pairs of wires, one in each direction (differential voltage principle). The IEEE standard specifies galvanic separation via a transformer for each RJ45 connection. This transformer protects the devices from damage caused by the line's high voltage and prevents voltage offsets caused by potential variations between the devices. The circuit diagram for a Gigabit Ethernet interface is shown in **Figure 1.**

Discrete Circuitry of the Gigabit Ethernet Interface

The Ethernet transformer (LAN transformer) is the device-to-Ethernet cable interface. The transformer provides the necessary galvanic isolation between the device and the cable while at the same time matching the impedance of internal logic and the balanced wire pairs. Furthermore, the transformer protects the device from transient interference and suppresses common mode signals between the transceiver IC and the cable, both within the device and between the external cable and the device's electronics. However, the device must also be capable of transmitting data at a rate of up to 1 Gbit/s without significantly degrading the transmit and receive signals. Additional components are needed to meet the matching and electromagnetic compatibility (EMC) criteria.

Figure 2 depicts a circuit diagram of the Gigabit Ethernet interface using discrete components. The LAN transformer provides DC isolation between the electronics and the network cable. The primary-side winding's middle tap depicts the so-called "Bob Smith" termination.

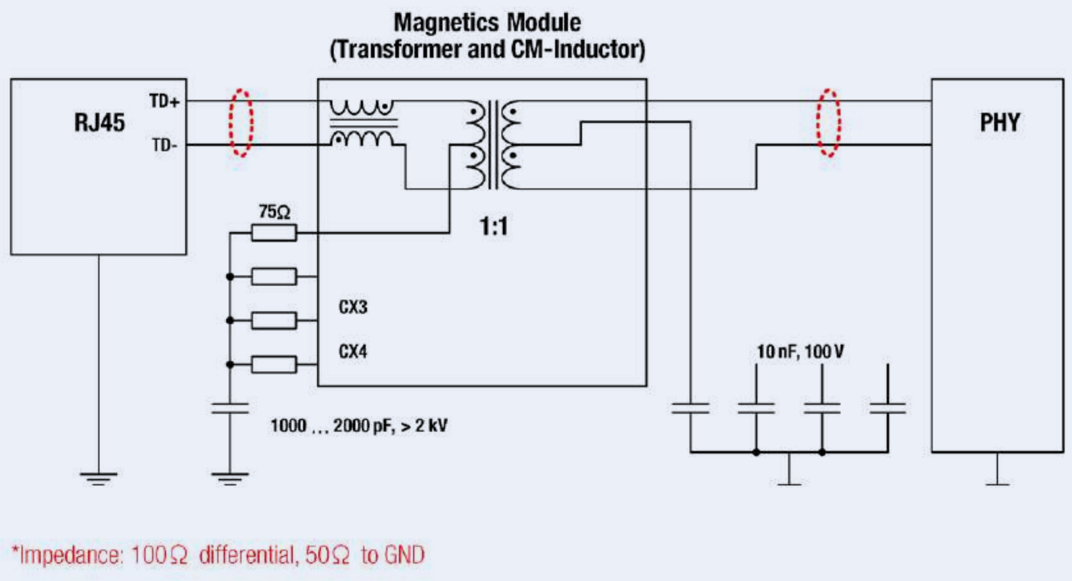


Figure 1: Basic circuit of a Gigabit Ethernet interface. Representation of a transmission channel with a total of four channels. (Source: Würth Elektronik)

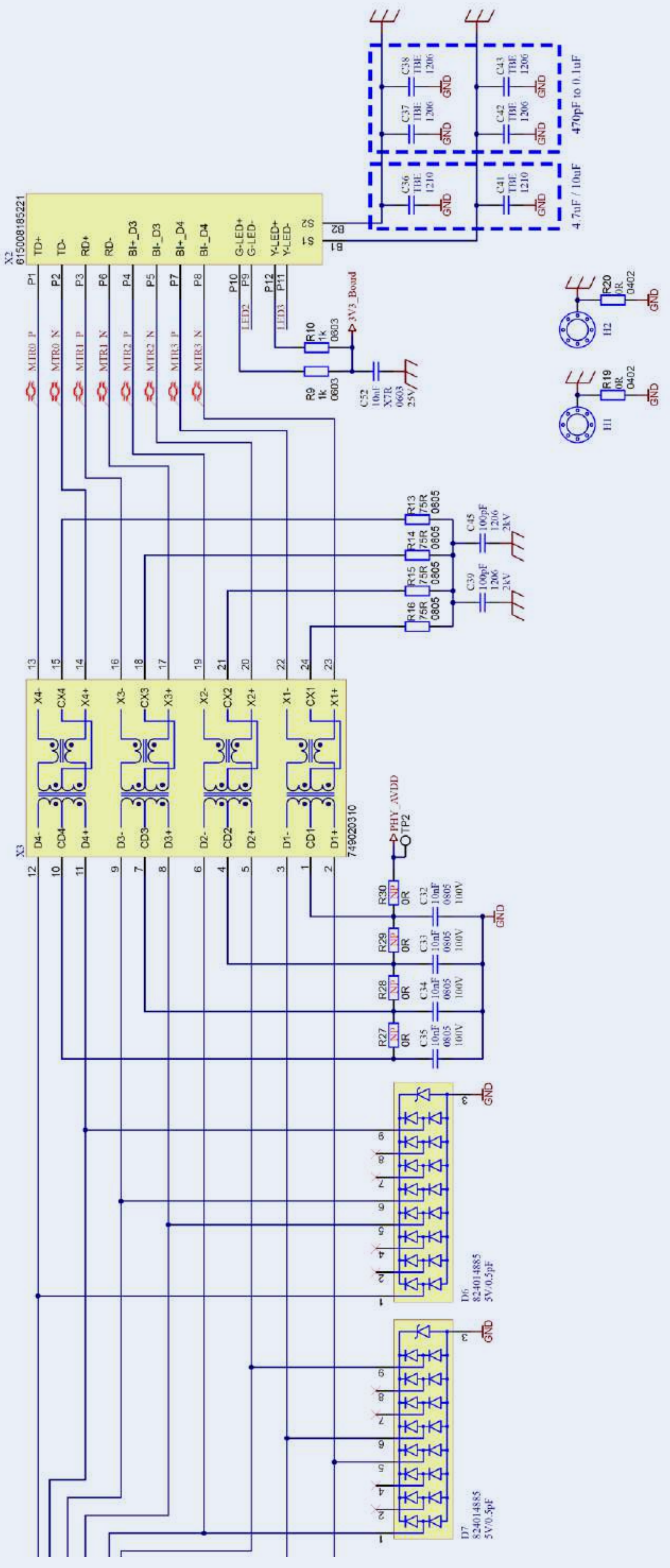


Figure 2: A Gigabit Ethernet interface's discrete circuit. Module X3 contains the LAN transformers and common-mode chokes necessary to prevent interference.

Here, one 75-Ω resistor is connected to each wire pair to form a "star point", which is then galvanically isolated and connected to the housing ground via two parallel 100-pF/2-kV capacitors. The additional common-mode chokes integrated in module X3 mitigate interference that is coupled both capacitively and inductively via the long Ethernet cables and could potentially impair Ethernet data communication as common-mode interference.

In Figure 2, R9, R10 and C52 are used to power the LEDs that are typically integrated in the connector socket. Through the capacitors C36 to C38 and C41 to C43, the shielding of the Ethernet socket can be connected to the board ground (GND). For sheet metal housings, it makes sense to omit these capacitors and connect the electronics' ground (GND) directly to the housing via screw connections. For plastic housings the capacitors should be fitted to connect the shield of the Ethernet cable to the reference ground. The 0-Ω resistors R19 and R20 have the same purpose. However, unlike with capacitors, there is no galvanic separation here. Alternative configurations were included in this section for "experimental" purposes to compare the shielding quality of various Ethernet cables. The capacitors C32 to C35 on the secondary side of the transformers connect the center taps of these in an RF manner to ground (GND). Galvanic isolation via capacitors is essential to eliminate DC equalization currents from the PHY. The resistors R27 to R30 are included to comply with some PHY manufacturers requirements (Current Mode Line Driver - Option) but are typically not needed if the PHY operates in "Standard Voltage Mode". However, the TVS diode arrays D6 and D7 are indispensable because they isolate transient interferences on the interface side of the PHY from the circuit ground (GND). On the secondary side, i.e. after the transformers of the X3 module, transient disturbances occur in common mode, and therefore a TVS diode must be connected to each terminal of the transformers against the reference ground. The secondary side of the transformer, however, has lower interference levels than the primary side. Important for the TVS diodes to function properly is a low-impedance connection of the diodes, on the one hand looped into the signal lines and on the other hand to ground.

Figure 3 illustrates the layout of all four layers of the board starting with the Ethernet interface area. The package/socket ground is isolated from the electronics GND in all four layers. Thus, the package ground's surfaces do not overlap with those of other layers to keep capacitive coupling as low as possible. The ground planes were plated through every 4 mm in a grid pattern. The Ethernet socket's signal lines are balanced, with a differential impedance of 100 Ω routed to the reference ground. The conductor pairs have a track width of 0.154 mm and are spaced 0.125 mm apart. The Ethernet socket is positioned on the PCB's edge to ensure a low-impedance connection to a metal enclosure if necessary.

The transformer module (X3) is placed nearby to minimize the effects of electrical coupling, or interference from long traces. As with the primary side, the secondary side of the transformer module must maintain a differential impedance of 100 Ω to the reference ground for the conductor paths. To avoid voltage drop due to parasitic inductance, the TVS arrays must be connected directly into the signal path and to GND.

EMC Compliance

In terms of electromagnetic compatibility (EMC), the board complies with industry standards for immunity (EN61000-6-2) and EN55032 Class B radio interference emission levels for multimedia equipment. Numerous factors must be considered when designing a 1 Gbit/s Ethernet interface. These include an RF-compatible circuit and layout design, a system-dependent ground concept, and the right choice of components. Only when all these factors are taken into account a product that functions reliably and meets stringent requirements can be developed. Further information on these topics, as well as on other interface standards, is available in various app notes published by Würth Elektronik at [1].

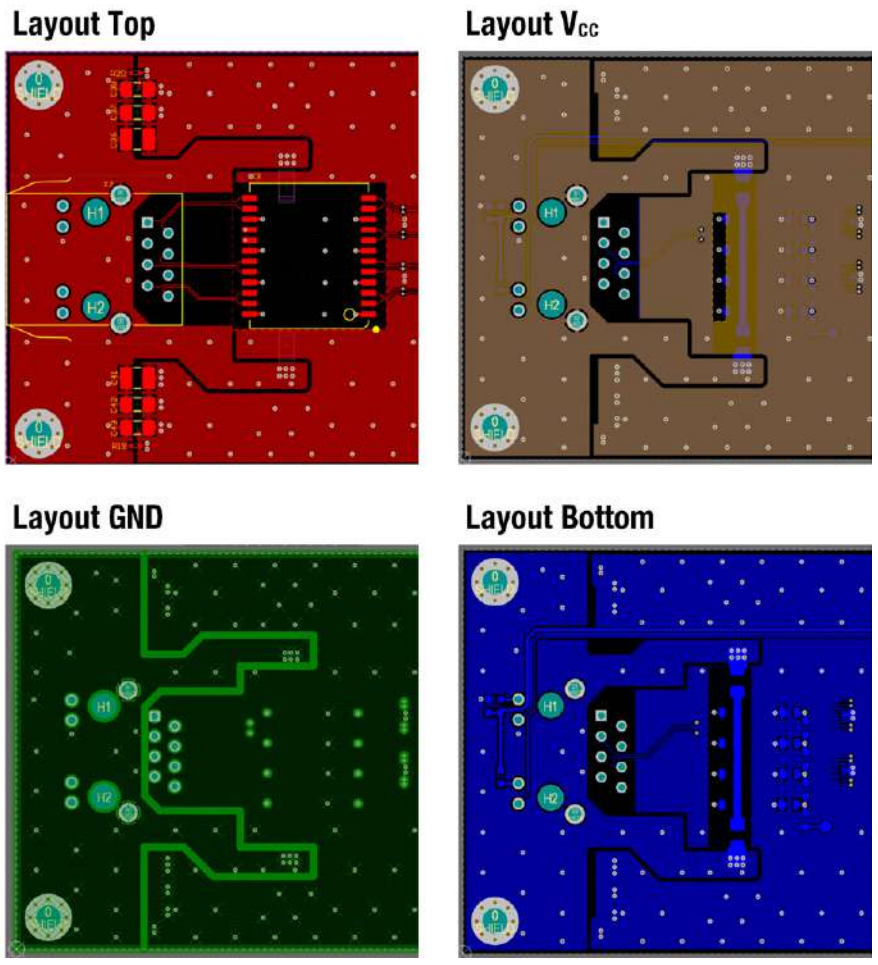


Figure 3: The layout of all four board layers of the Ethernet interface area.

220182-01

About the Author

Dr.-Ing. Heinz Zenkner is a freelance consultant at Würth Elektronik in the areas of technical marketing and application engineering as well as a lecturer at the technical academy in the area of EMC. At the same time, Heinz is a publicly appointed and sworn EMC expert. He has authored numerous technical journals and books, and has worked as a lecturer at various universities, the IHK and at numerous seminars.

WEB LINKS

[1] Würth Elektronik Application Guide: https://www.we-online.com/applicationguide/en

Bringing Real-Time Object Detection to MCUs with Edge Impulse FOMO

By Jan Jongboom, Edge Impulse

We humans rely heavily on sight to perform many daily tasks, from some of the most basic to the most complex. With one look, we know if there are people in the room with us, if there's an elephant nearby, or how many free parking spaces are available. Despite the importance of vision, though, many embedded devices still can't perceive things visually. Wouldn't it be amazing if we could teach *all our devices* to see the world the way we do?

In recent years, there have been some amazing developments in computer vision, fueling progress in things like self-driving cars and biometric immigration gates (very useful if, like me, you travel a lot!). But these use cases are incredibly computationally expensive, requiring costly GPUs or special accelerators to run.

The awesome thing is that not all computer-vision tasks require such intensive compute. Any yes/no question ("Do I see an elephant?," "Is this label properly attached to the bottle?") can add tremendous value to constrained embedded devices. What's more, these problems of image classification can even be solved by today's microcontrollers.

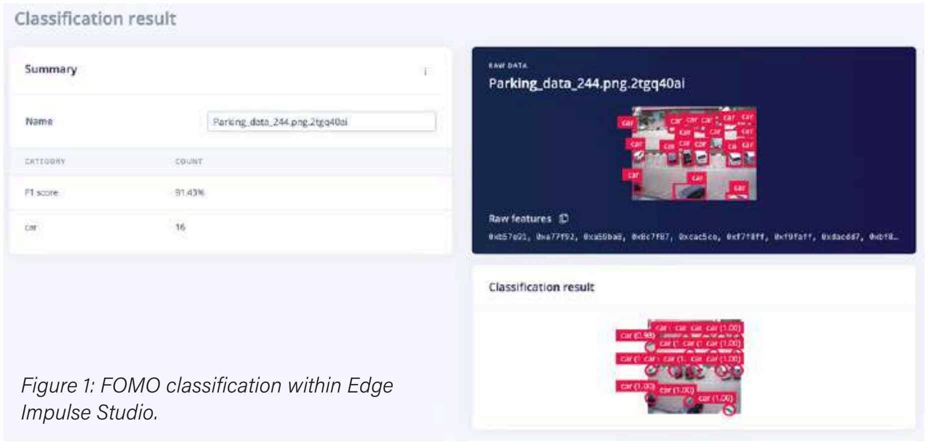
Imagine if we could add even more advanced vision capabilities to every embedded device!

Say Hello to FOMO

We're making it a reality. We developed a novel neural network architecture for object detection called Faster Objects, More Objects, or FOMO (**Figure 1**). It's designed from the ground up to run in real-time on microcontrollers, so embedded engineers can (ahem) avoid the fear of missing out when it comes to computer vision.

Fast, Lean & Flexible

FOMO is capable of running on a 32-bit MCU, like an Arm Cortex-M7, with a frame rate of 30 frames per second. And the next time you choose a Raspberry Pi 4 type device, you'll be able to do object detection at a rate of about 60 frames a second. That's roughly 30 times faster than MobileNet SSD or YOLOv5.



Classification result

Summary

Name: Parking_data_244.png.2tgq40ai

CATEGORY	COUNT
F1 score	91.43%
car	16

RAW DATA

Parking_data_244.png.2tgq40ai

Raw features

```
0xb57e01, 0xa77f92, 0xa59ba8, 0x8c7f87, 0xcac5c0, 0xf7f8ff, 0xf9faff, 0xdacdd7, 0xb18…
```

Classification result

Figure 1: FOMO classification within Edge Impulse Studio.

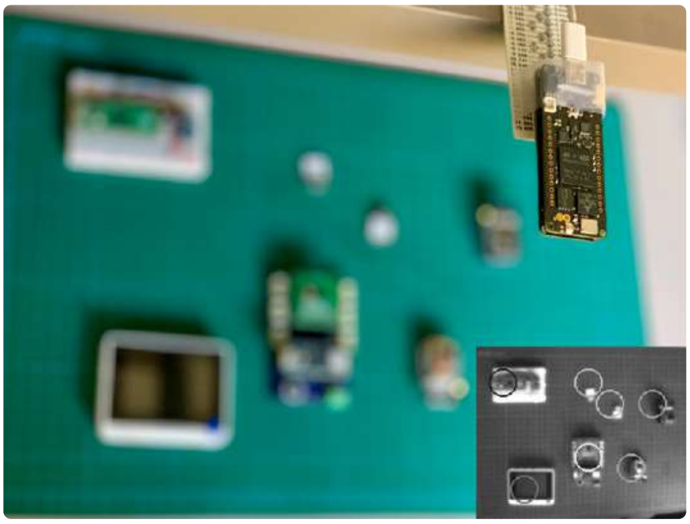


Figure 2: Run object detection on a wide variety of dev boards, including the Arduino Portenta.

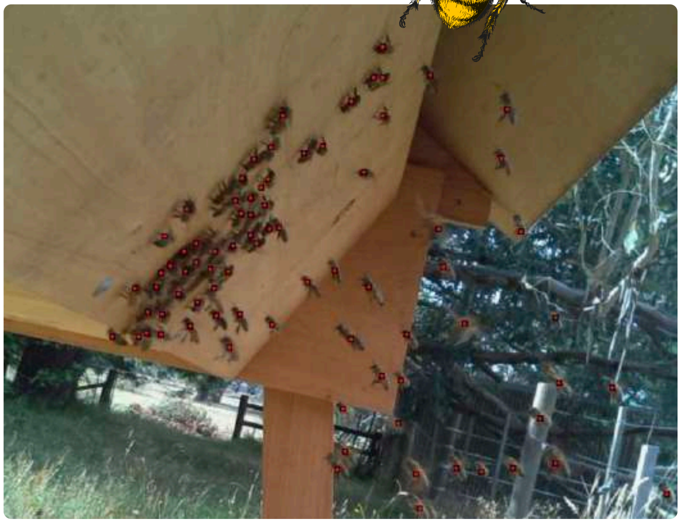


Figure 3: Here's a former iteration of the FOMO approach used to count individual bees.

FOMO scales down to about 100 kilobytes in RAM, making it possible to run object detection in real-time on everything from highly-constrained Arm Cortex-M4 cores to more powerful ones, like the Cortex-M7 cores on the Arduino Portenta H7 (**Figure 2**), the new Arduino Nicla Vision (another dual Arm Cortex-M7/M4 CPU), or even specialized DSPs such as the Himax WE-I.

FOMO can scale from the tiniest microcontrollers all the way to full gateways or GPUs. This high degree of flexibility also makes FOMO useful when fault detection requires identifying variations that are very, very small within an image.

In an MCU with strictly limited compute and memory capacity, it's best to use an image size of about 96x96 pixels. But with a larger microcontroller device, 160x160 pixels is probably fine. The important thing is that FOMO is fully convolutional, so it works on any arbitrary input size. If you need higher granularity, more detail, or more objects, you can just scale up the input resolution.

It Sees the Little Stuff

As long as the objects in the frame are of similar size and don't overlap, this new architecture can even spot and count lots of very small objects very effectively (**Figure 3**). That's something that MobileNet SSD and YOLOv5, despite being larger and more capable models, can't do very well.

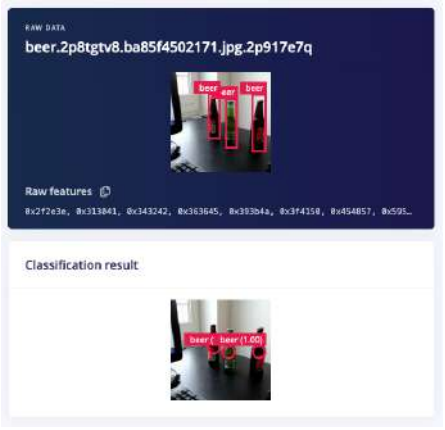
No More Missing Out

FOMO is available today, runs on a wide variety of computing platforms, and is compatible with Linux systems, Cortex-M microcontrollers, and specialized DSPs. Add a camera and Edge Impulse, and you're all set.

With FOMO, you can quickly add object detection to just about any camera-based device, and avoid the fear of missing out that, until now, embedded engineers have had to deal with when it came to computer vision (**Figure 4**).

To learn more about FOMO and experiment with your own algorithm, visit **edgeimpulse.com/fomo**. ◀

220207-01



beer.2p8tgtv8.ba85f4502171.jpg.2p917e7q

```
0x2f2e3e, 0x313841, 0x343242, 0x363645, 0x393b4a, 0x3f4150, 0x454857, 0x595…
```

Figure 4: Training on the centroids of beer bottles. On top the source labels, at the bottom the inference result.

About the Author

Jan Jongboom is an embedded engineer and machine learning advocate, always looking for ways to gather more intelligence from the real world. He has shipped devices, worked on the latest network tech, simulated microcontrollers and there's even a monument in San Francisco with his name on it. Currently he serves as the cofounder and CTO of Edge Impulse, the leading development platform for embedded machine learning with 80,000+ projects.

Traveling Wave Tubes

Peculiar Parts, the Series

By **Neil Gruending**

The world of RF amplifiers is fascinating because of the wide range of methods in use. Today, many are constructed from solid-state components, but there are still situations where vacuum tubes are the only suitable choice. We've looked at klystrons in the past, so, this time, let's look at the traveling-wave tube amplifier, another unsung electronic hero.

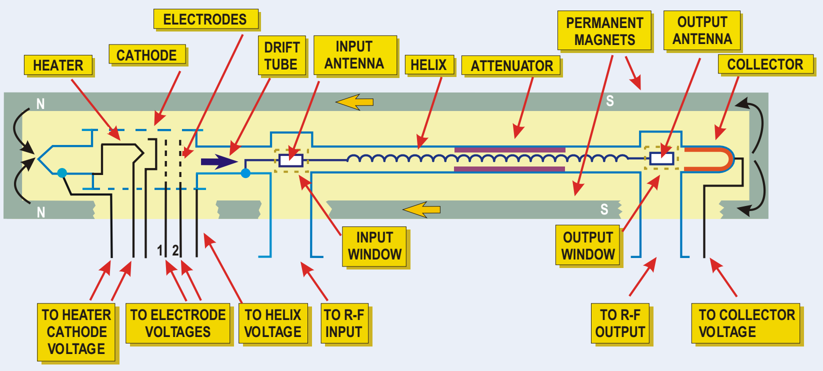


Figure 1: Basic design of a traveling-wave tube.

One of the most fascinating things about traveling-wave tube amplifiers is how they work. They feature a heater, cathode, and acceleration electrodes to form an electron gun, much like a cathode-ray tube, that beams a stream of electrons to the collector (**Figure 1**). This stream is focused by an external magnetic field that is usually made of permanent magnets. Using velocity modulation, which mixes the electron stream with the incoming RF electrons, the tube amplifies the applied RF input signal.

Since the streaming electrons travel much more slowly than the RF electrons, the RF signal is fed through a spiral wound wire, called the helix. This slows the RF signal down to match that of the electron stream.

As the RF electrons proceed down the helix, they modulate the velocity of the electrons in the stream because the in-phase electrons speed up, and the out-of-phase electrons slow down. These modulated electrons then bunch together, inducing an amplified signal back into the helix that is then picked off the end of the helix using a directional coupler.

When compared to klystrons [1], traveling-wave tubes have the advantage of wider bandwidths. Additionally, they don't require resonant components, making them ideal for lower power microwave applications like radar or even spacecraft and satellites. A great example is the Collins Radio S-Band amplifier (**Figure 2** and **Figure 3**) used in the Apollo space program [2]. It was a compact, 32-pound, 20-W amplifier that transmitted all of the voice, data, and television back to NASA's network of 26-m, earth-based dish antennas. By comparison, the ground station used a focused 10,000-W signal to communicate back to the craft.

Even though traveling-wave tubes are mostly the domain of commercial applications, a small group of enthusiasts still like experimenting with these wonderful little amplifiers in amateur microwave transmitters [3]. Their biggest challenge, however, is finding the tubes!

210418-01

Questions or Comments?

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

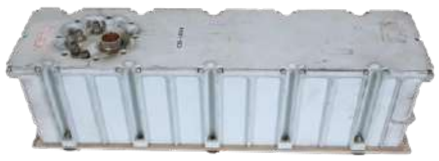


Figure 2: Collins Radio S-Band traveling-wave tube amplifier used for communication with earth during the Apollo mission. (Source: Ken Shirriff)

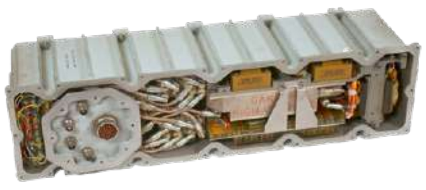


Figure 3: Operating at several thousand volts, the Collins Radio amplifier was a tightly-packed tangle of coaxial cables. (Source: Ken Shirriff)

WEB LINKS

[1] N. Gruending, "Klystrons, Weird Component # 12," Elektor 3/2015: https://bit.ly/2UW4k9G
[2] K. Shirriff, "Inside a 20-Watt Traveling-Wave Tube Amplifier from Apollo," Ken Shirriff's Blog, July 2021: https://bit.ly/3ea8lOn
[3] H. Griffiths, "Travelling Wave Tube Amplifiers," The National Valve Museum, September 1980: https://bit.ly/3wA8aCn

Narrowband Internet of Things

source: shutterstock.com

Standards, Coverage, Agreements and Modules

By **Tam Hanna** (Slovakia)

Curious about Narrowband Internet of Things (NB-IoT)? Is it for you? Let's take a look.

Along with LoRa and Sigfox, mobile communication networks are also a good option for the transmission of IoT sensor data. The upgrade from EDGE to UMTS made this option even more attractive, since using a faster transmission system can in the end be better than using a lower-power but slower system. However, the immense bandwidth and power hunger of 4G/LTE make this rule of thumb a bit less relevant. The power consumption of the transmitters is significantly higher, and on top of that the modules are more expensive. Nevertheless, it can be worthwhile.

In the framework of the specification *3GPP Release 13*, designated by LTE as 'informational', the GSM Association defines two systems for the Internet of Things. The first is Narrowband Internet of Things (NB-IoT), and the second is LTE-M, also known as LTE Cat-M1 or eMTC.

LTE-M is basically a 'light' version of LTE (4G) with a bandwidth of 1.4 MHz, while NB-IoT is a dedicated wireless communication standard for the Internet of Things. The key difference is that LTE-M additionally supports voice transmission with VoLTE, while an NB-IoT system exclusively transmits data messages.

The NB-IoT channels, each with a width of only 180 kHz, use a subset of the methods implemented in the full version of LTE. Uplink uses a simple version of the frequency division multiple access (FDMA) method, while downlink uses orthogonal FDMA (OFDMA). The quadrature phase shift keying (QPSK) modulation method does not require especially complex hardware in terms of processing power.

However, it should be noted that introducing NB-IoT usually incurs

Table 1: Frequency bands.

Region	Bands
Europe	3, 8, 20
(Former) CIS countries	3, 8, 20
North America	2, 4, 5, 12, 66, 71, 26
Asia Pacific (APAC)	1, 3, 5, 8, 18, 20, 26, 28
Sub-Saharan Africa	3, 8
Middle East and parts of North America	8,20
Latin America	2, 3, 5, 29

additional costs for the carrier for the new hardware. Due to the extremely narrow bandwidth, NB-IoT can easily fit into the guard band surrounding the LTE frequency packets. On the other hand, using NB-IoT in stand-alone mode is of course also possible.

Looking at Performance

Even the technically most attractive wireless communication standard is of no use if the transmission capability is insufficient for the intended task. In the case of NB-IoT, the version is an important consideration because there are differences between LTE Cat NB1 (Release 13) and LTE Cat NB2 (Release 14). The older version can only achieve 26 kbit/s in upstream, but Cat NB2 is significantly faster with 127 kbit/s upstream and 159 kbit/s downstream. For comparison, conventional (not HSDPA) 3G initially achieved 380 kbit/s. LTE Cat M1 currently runs at around 1 Mbit/s upstream and downstream, and Release 14 raises this to 4 Mbit/s upstream and 7 Mbit/s downstream.

The differences in latency times are enormous. LTE-M can usually achieve 15 ms, while with NB-IoT the recommended 'working range' is from 1.6 s to as much as 10 s. The module manufacturer Sierra Wireless, especially popular in the USA, describes the situation as follows:

"Another important fact to consider is that there are no NB-IoT use cases that LTE-M can't also support. In other words, LTE-M supports any LPWA application, whereas NB-IoT is designed for simpler static sensor type applications."[1]

In addition, only version 2 of the NB-IoT standard supports position data provision by the network operator. If the module does not have GPS capability or you want to do without an external antenna, you can use this approach to obtain basic position data. Release 14 also accelerates searching for new cells, which is mainly beneficial for moving devices. Despite these new benefits of Cat NB2, LTE-M is still the best choice for automotive and other mobile applications because it provides smarter cell handover. The final improvement concerns transmit power: Super Low Power transmitters [2], which can operate with only 14 dBm, are only allowed in Release 14.

If at some point in time you got your hands on a 4G module for Verizon, the natural question here is which bands are used. Band 13, which is only important for North America, has caused problems for many Asian or European module providers. **Table 1** is taken from the *Deployment Guide* [3] of the GSM Association. You should make sure that the module you choose supports all the bands that are used by your preferred carrier.

Availability and Agreements

It goes without saying that wireless communication standards are only worthwhile if they are also available in practice. In the case of the two IoT wireless communication standards, you should have a look at the interactive world map of the GSM Association in **Figure 1** [3] (status as of September 2021). As you can see, Mexico is the only country where only CAT-M is available (probably because of the larger range), while "NB-IoT only" is more widely available in the rural areas of Asian countries and, remarkably enough, in Eastern Europe. In the highly industrialised regions of Europe, North America, Asia, Australia and Oceania, both versions are available.

CAT-M agreements are generally ordinary agreements in which the total usage volume and the number of SIM cards determine the overall cost. For the sake of completeness, it should be noted that with regard to cost, an IoT provider such as PodGroup is often a better choice than a prepaid SIM card purchased on the open market.

The claim that NB-IoT is not subject to duty cycle restrictions is not borne out by the author's practical experience as a consultant. Talking with your mobile provider about IoT connectivity is and will remain a matter of negotiation, and all too often limits on the number of packets in a given time interval will be imposed. Operators rarely publish their exact conditions in this regard, which makes the following statement from T-Mobile USA all the more remarkable:

"Join the first nationwide NB-IoT network to power asset tracking, connected cities, and more. Limited time offer; subject to change. Taxes and fees may be additional. Plan includes 10 single-packet transactions per hour at up to 64 Kbps, up to 12 MB. Full service payment due at activation."[4]

Interestingly, this is only an individual opinion, and Hutchison Holding Ltd has confirmed that the total volume of data traffic (within the bounds of the agreed amount) can be used up in one day. Tom Tesch, the Austrian spokesperson of Hutchinson, says in this regard:

"The data rate of NB-IoT — in accordance with the standard — is very low and primarily suitable for the transmission of individual measurements or status values. For this reason, more than 5 to 10 MB per month is very rarely needed for NB-IoT devices. For more bandwidth-intensive applications, such as the transmission of photos or videos, 3G/4G and of course 5G are more suitable technologies. There are currently no limits on when the volume can or may be used, which means that the entire volume can also be used up in one day."

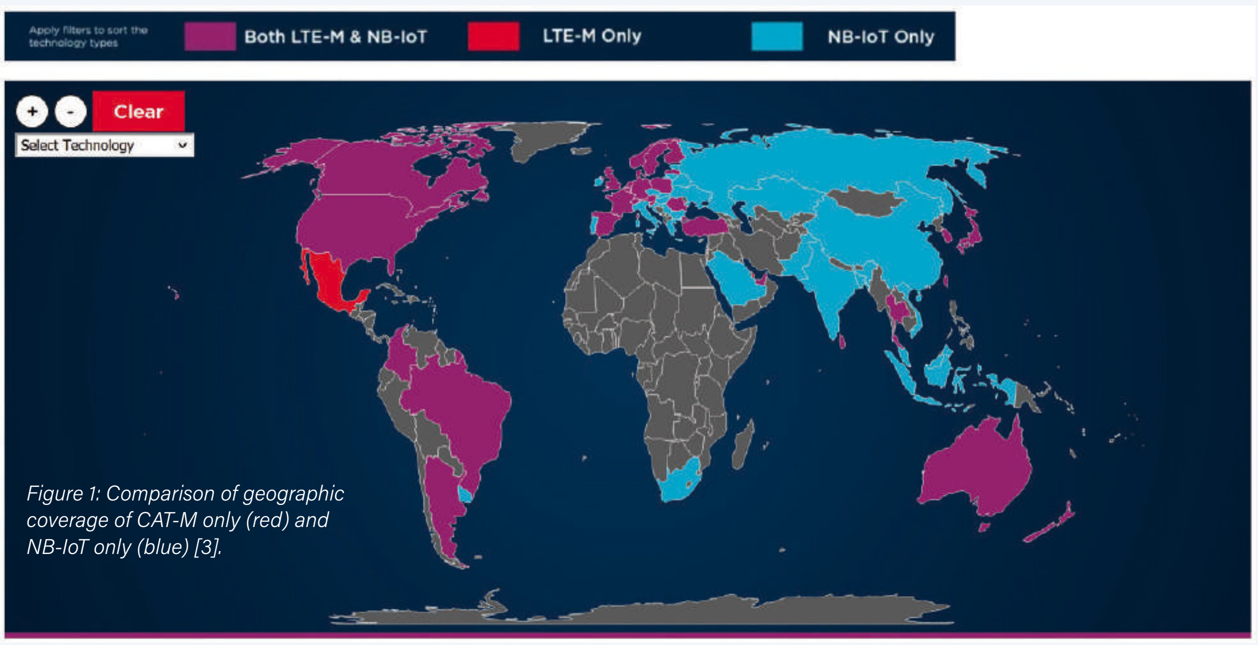


Figure 1: Comparison of geographic coverage of CAT-M only (red) and NB-IoT only (blue) [3].

How to Get Started

After these basic considerations, it's time to start thinking about how you can integrate NB-IoT into practical systems. Of course, development of customer-specific modems is not feasible for most companies, but in the past we have described the 'design in' process for wireless modules in detail, for example in *Elektor* 5-6/2021 [5].

If you don't want to start developing your own board right away, one option is to use a 'turnkey' evaluation board — although the availability of Qualcomm ICs is proving to be a problem in this regard.

Two possibilities are the NBIOT-BG96-SHIELD from Avnet, which integrates a Quectel BG96 module, and the 5G NB IoT click board from MikroElektronika, which hosts a Cinterion module. Arduino also offers a small development board in the form of the MKR NB 1500. However, both of these boards cost more than 50 dollars.

In many cases, it is no longer permitted to supply evaluation board equipped with SIMs, so a massive rollout of applications based on NB-IoT is far from easy. The reason for this is that network operators have not yet packaged the technology for end users. This is also openly admitted by operators, as illustrated by the following statement from Hutchison:

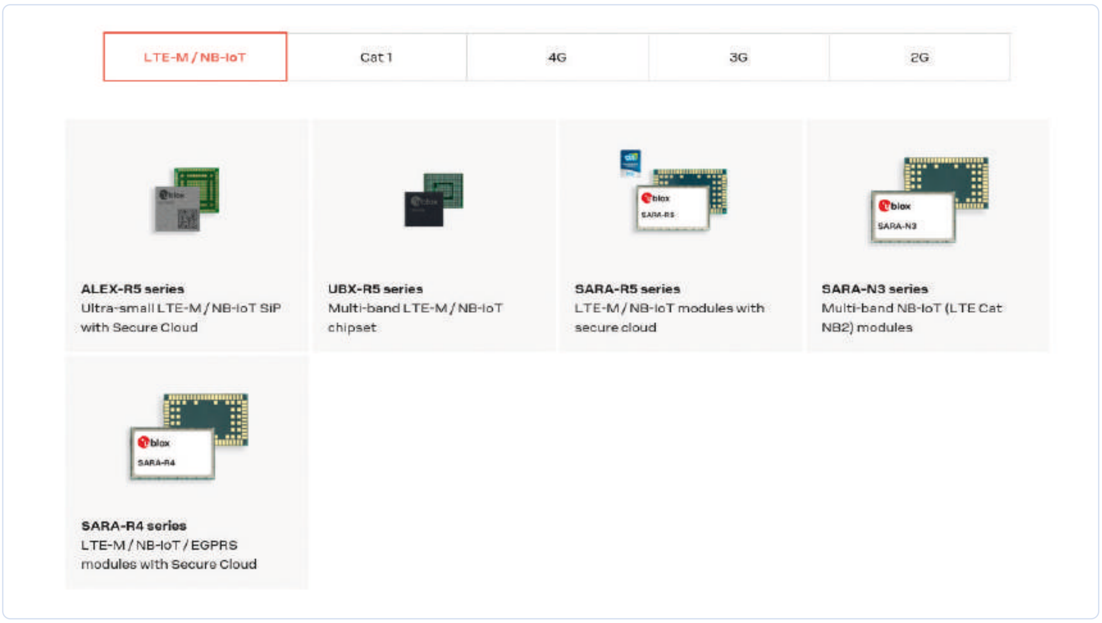
"NB-IoT is a very young and innovative network. As there are hardly any devices available on the market, the target group primarily consists of business customers in the hardware (and software) development environment. This means that our offering is currently exclusively oriented to business customers, for which we create a tailored offer in the course of a consultation process."

When working with 'ordinary' 2G/3G/4G systems, one way to get around this is to use a 'virtual' mobile communication provider such as PodGroup. When asked about this, they answered that NB-IoT is currently not really suitable, especially for 'global' solutions that need to work with a single SIM card.

There are two reasons for this. Firstly, that the NB-IoT rollout is still relatively limited. And secondly, that roaming agreements between the different network operators have generally not yet been adapted to the new NB-IoT wireless communication standard. Liked tax treaties between countries, it takes a long time to achieve such adaptations. In short: international NB-IoT roaming is still in its infancy.

Is It Worthwhile?

Searching for a practical module that supports only NB-IoT is certainly a very tricky endeavour. Quectel, for example, offers two versions even with the smallest series (BC660): one with only NB-IoT, and the other with both eMTC and NB-IoT. Both wireless standards are also present in larger families, such as the very popular BG95 and BG96. Open market prices for these modules can only be found at SOS Electronic: the BC660K-GL costs €7.63 is small quantities, while the version with LTE-M and NB-IoT is not listed. The price there for the BG96 is €19.


LTE-M / NB-IoT | Cat 1 | 4G | 3G | 2G

ALEX-R5 series
Ultra-small LTE-M / NB-IoT SiP with Secure Cloud

UBX-R5 series
Multi-band LTE-M / NB-IoT chipset

SARA-R5 series
LTE-M / NB-IoT modules with secure cloud

SARA-N3 series
Multi-band NB-IoT (LTE Cat NB2) modules

SARA-R4 series
LTE-M / NB-IoT / EGPRS modules with Secure Cloud

Figure 2: Searching for a pure CAT-M module doesn't turn up many results [6].

Figure 3: A traditional Stierian windmill serving as a scarecrow. (Source: Martin Geisler, CC BY-SA 4.0 [7]).

A search for u-blox [6] yields more results. The SARA-N3 family includes a module exclusively intended for the NB-IoT set of protocols, but the Swiss company does not offer a pure CAT-M device (see **Figure 2**).

At Gemalto, whose takeover by Thales has made the website even more confusing than it used to be, there is a pure CAT-M module in the form of the EMS31, along with a pure NB-IoT module (ENS22) with the same form factor. At the Czech distributor Sectron you can compare prices: the EMS31 costs €14, the ENS22 only €8.

Information on current consumption (in roundabout form) can be found in the data sheets, which go by the name 'Hardware Interface Description'. The highest current consumption of the EMS31 occurs when operating in Band 4 and is 239 mA with a supply voltage of 3.8 V. For the ENS22, the highest current listed is 404 mA in Band 28, but it should also be noted that wireless modules often require peak currents like this only for a very short time.

WEB LINKS

[1] LTE-M vs. NB-IoT: What are the Differences?: https://www.sierrawireless.com/iot-blog/lte-m-vs-nb-iot/
[2] Wikipedia entry on narrowband IoT: https://en.wikipedia.org/wiki/Narrowband_IoT
[3] GSMA: Word map of IoT wireless communication standards: https://www.gsma.com/iot/deployment-map/
[4] T-Mobile Narrowband IoT web page: https://t-mo.co/3EC5Jo4
[5] Tom Hanna, "Do Not Fear the Cellular Module!," Elektor Mag 5-6/2021: https://www.elektormagazine.com/magazine/elektor-175/59527
[6] u-blox mobile communication modules: https://www.u-blox.com/en/cellular-modules
[7] Klapotetz: https://bit.ly/3nOr0Fb

What's In It for You?

From a technical point of view, NB-IoT works perfectly, and once you have arranged an agreement with a carrier, the effort for network operation is limited to a phone call to your lawyer, unlike the situation with a home LoRa WAN. The relatively low peak and quiescent current consumption of the modules also helps to keep your electricity bill within bounds.

Whether or not it's worthwhile in the end is primarily a matter of scale, just like tax havens such as Dubai or Monaco. If you buy five modems in a year, operate with a 'full' 4G module or, even better, a module with a beefier power adapter and costing a few euros more, the sad experience of the author is that in practice you will repeatedly need the 'other' wireless standard, if only because some base stations do not support every wireless communication standard.

Naturally, the situation looks different if you are purchasing 50,000 modems that will all go to the same customer. If the mayor's office of Großdorf am Klapotetz (**Figure 3**) needs NB-IoT, the local carrier will probably upgrade their network, and the cost savings from the large number of devices will also help.

180021-01

Questions or Comments?

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

Contributors

Text: Tam Hanna
Editor: Rolf Gerstendorf
Translation: Kenneth Cox
Layout: Giel Dols

Advertisement

Ensure your products are 100% authentical

Mouser was the first SAE AS6496 accredited distributor

Widest selection of electronic components in stock™
mouser.co.uk/authentic

MOUSER ELECTRONICS

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**).

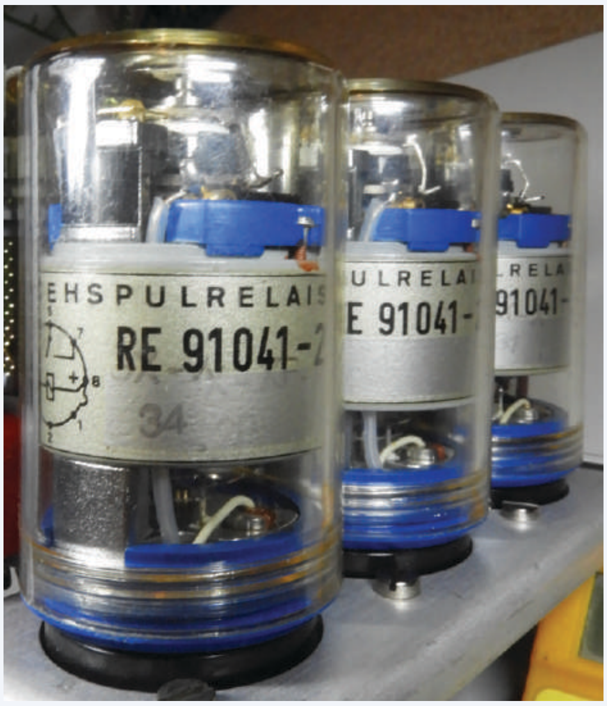


Figure 1: The relays, shown here from behind, with their type number and plastic case on display.

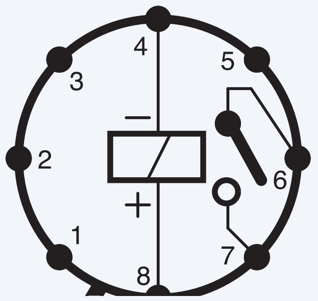


Figure 2: This circuit diagram is printed on the side of the relay.

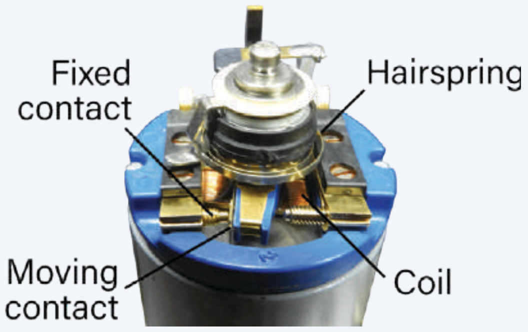


Figure 3: An annotated photo highlighting the location of the hairspring, coil, and contacts.

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 kΩ 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 µA 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.

Advertisement

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

Dragino LPS8 Indoor Gateway

Speedy LoRaWAN Gateway Setup

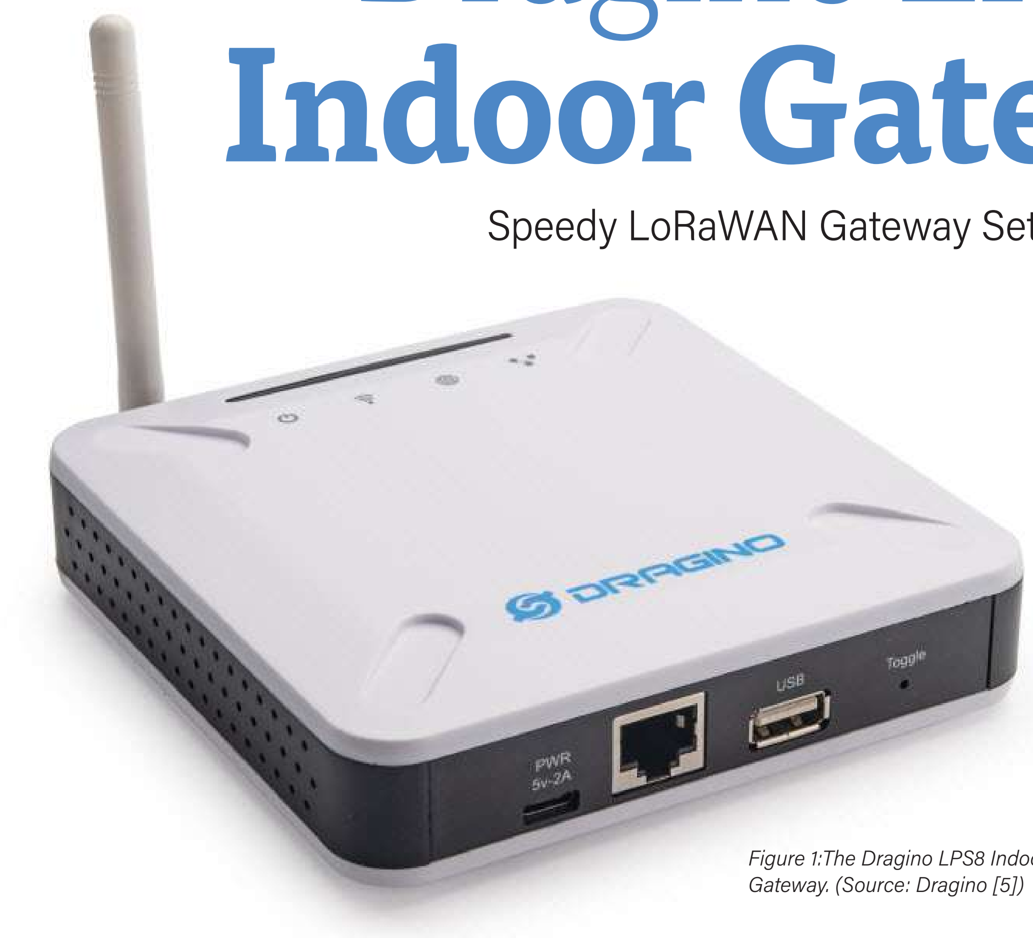


Figure 1:The Dragino LPS8 Indoor Gateway. (Source: Dragino [5])

By Mathias Claußen (Elektor)

We have often described how you can interconnect your own electronics devices using a LoRaWAN link. If you are not within range of an existing LoRaWAN gateway, or if you simply want to delve a bit deeper into the topic, you can set up and operate your own gateway. We tried this using the low-cost Dragino LPS8 Indoor Gateway.

LoRaWAN is a topic we have featured many number times in Elektor. It is relatively easy to build a basic LoRaWAN node which has an associated sensor or actuator module. In this type of setup, a LoRaWAN module (which handles communication to the network) is connected to a microcontroller board such as an STMicroelectronics STM32 or Raspberry Pi Pico [1, 2], and this provides an interface to the sensor. In order for the data sent to and from the node via LoRa to be transported further, a remote station is required. In this case, a LoRaWAN gateway will accept the data over the air via LoRa and forward it to an Internet platform like The Things Network (TTN). You can use a pre-existing gateway that's already been set up in your area (many are run by volunteers), or you can set up your own gateway. I have been using a Dragino LPS8 to provide an Indoor Gateway for more than a year now.

The Dragino LPS8

The Dragino LPS8 Indoor Gateway (**Figure 1**) is housed in a plastic enclosure and could easily be mistaken for a Wi-Fi router. The electronics inside are powered by a small Atheros (today Qualcomm) AR9331 Wi-Fi SoC clocked at 400 MHz which is specifically designed for use in router platforms and access points. With 64-MB RAM and 16-MB Flash, its processing power is not spectacular when compared with something like a Raspberry Pi Zero 2 W, but it is more than enough for the functions that the gateway needs to perform. The SoC also supports Wi-Fi according to 802.11 b/g/n and provides a 10/100 Mbit LAN port. The communication rates available are more than capable of handling the relatively slow data rate used by LoRaWAN. The gateway itself does not have to provide a lot of computing power either, as it only takes care of the integrated LoRa

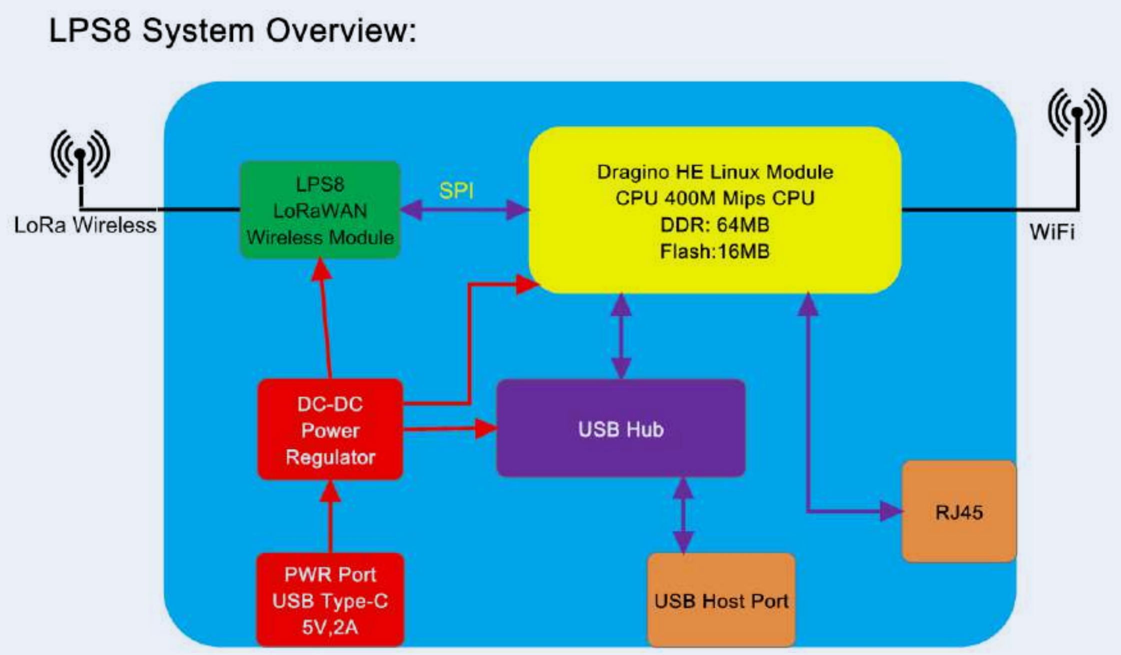

LPS8 System Overview:
LoRa Wireless
LPS8 LoRaWAN Wireless Module
SPI
Dragino HE Linux Module CPU 400M Mips CPU DDR: 64MB Flash:16MB
WiFi
DC-DC Power Regulator
USB Hub
RJ45
PWR Port USB Type-C 5V,2A
USB Host Port

Figure 2: Block diagram of the Dragino LPS8. (Source: Dragino [6])

transceiver module and forwards the data to the Internet. A block diagram can be seen in **Figure 2**.

The LoRa transceiver is a combination of a Semtech SX1308 LoRa baseband chip (**Figure 3**) and two SX1257 front-end modules (**Figure 4**). This combination provides the conversion from the radio interface to Ethernet. The gateway is powered via its USB type C port and requires a 5 V/2 A (10 W) mains adapter.

As the name of the gateway suggests, the device is not weatherproof and is intended for use inside a building, so the environment should be dry and relatively dust-free. The building structure and internal walls will reduce the radio coverage compared to an equivalent device mounted outdoors in free space with a mast-mounted antenna.

The LPS8 Manual, Firmware and Setup

The most recent version of the Dragino manual (available online [3]) describes how to set up the gateway. The manual has been continuously maintained since the product was released and reflects the features and updates of the current firmware. This is indeed praiseworthy; I just wish that some other product manufacturers would adopt the same attention to detail when it comes to documentation.

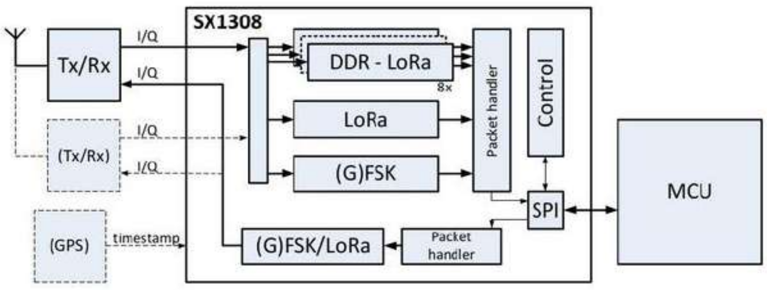


Figure 3: Block diagram of the SX1308 baseband chip. (Source: Semtech [7])

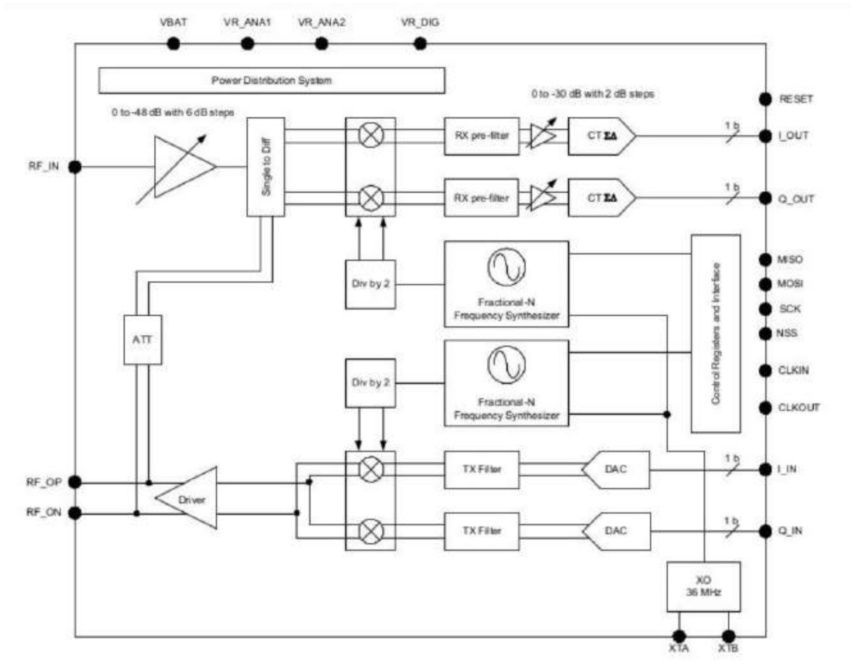


Figure 4: Block diagram of the SX1257 frontend module. (Source: Semtech [8])

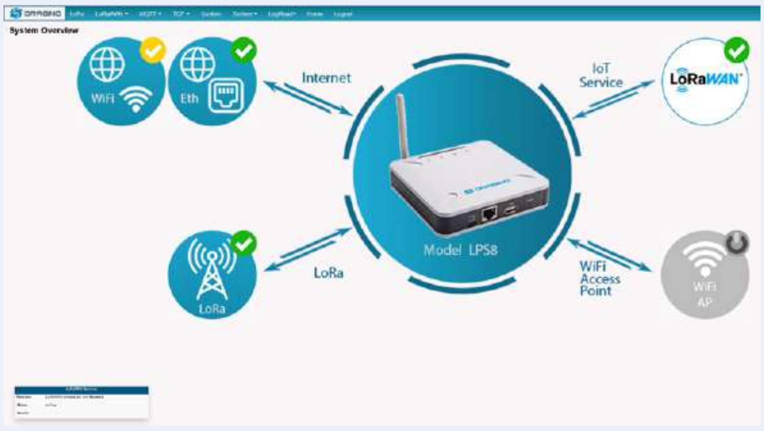


Figure 5: Communication paths available via the LPS8 gateway.

The firmware itself is also well maintained. The current release is dated November 4, 2021 (as of December 15, 2021) [4]. It's advisable to update to the latest version before the gateway is put into service. That will ensure any known bugs or security weaknesses should be ironed out as far as possible.

The manual guides you through the setup. All you need to do is configure the network appropriately and make the settings for the LoRaWAN link (e.g., The Things Network). From this point, the LoRaWAN gateway is ready for use (**Figure 5**).

An OpenWRT Substructure

Even though the first page of the web interface doesn't suggest it, the Linux-based Open Wireless Router (OpenWRT) firmware is used as the basis for the Dragino LPS8 indoor gateway. This not only takes care of the LoRaWAN gateway function, but also provides a number of other settings for the router (IP addresses, forwarding, Wi-Fi).

Thanks to the OpenWRT substructure, an LTE or 5G modem can also be connected to the gateway USB port if no other link to the Internet is possible at the device's location. If you like, you can also access the Linux command line using SSH. (Do so at your own risk!) Additional packages can be installed via the web interface or command line to add more functions to the device.

A Reliable Solution

I have personally been using a Dragino LPS8 now for over a year. During that time, it has proved to be a low-maintenance and reliable LoRaWAN gateway, which is really all you could ask of such a device. It continues to do a good job servicing my various LoRaWAN nodes and gives excellent coverage throughout the building (and surrounding area). If you are thinking of installing an inexpensive LoRaWAN gateway in a domestic environment, you should take a closer look at the Dragino LPS8 Indoor Gateway, which is currently available from the Elektor Store [5].

210680-01

Questions or Comments?

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

Contributors

Text: **Mathias Claußen**
Editor: **Jens Nickel, C.J. Abate**
Translator: **Martin Cooke**
Layout: **Harmen Heida**

RELATED PRODUCTS

- **Dragino LPS8 Indoor LoRaWAN Gateway (868 MHz) (SKU 19094)**
  www.elektor.com/19094
- **Seeed Studio LoRa-E5 STM32WLE5JC Development Kit (SKU 19956)**
  www.elektor.com/19956

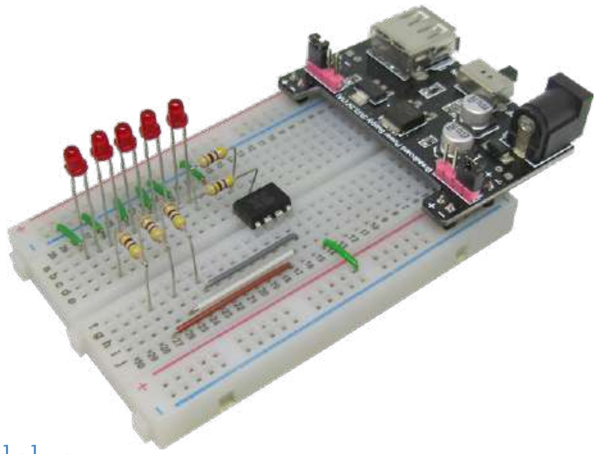
WEB LINKS

[1] M. Claußen, "My First LoRaWAN," ElektorMag 3-4/2020: http://www.elektormagazine.com/magazine/elektor-141/57159
[2] M. Claußen, "LoRa with the Raspberry Pi Pico," ElektorMag 7-8/2020: http://www.elektormagazine.com/magazine/elektor-179/59721
[3] Dragino LPS8 Indoor Gateway Handbook: http://www.dragino.com/downloads/index.php?dir=LoRa_Gateway/LPS8/
[4] Firmware download of Dragino LPS8 Indoor Gateway: https://bit.ly/LPS8-firmware-release
[5] Gateway picture resource: http://www.dragino.com/media/k2/galleries/148/LPS8-10.jpg
[6] Dragino LPS8 Indoor Gateway Manual: https://bit.ly/LPS8-user-manual
[7] The Semtech SX1257 Front-End Data sheet: https://sforce.co/3fZmy1f
[8] The Semtech SX1308 Transceiver Data sheet: https://sforce.co/32zxAqV

Explore ATtiny Microcontrollers Using C and Assembly Language

Sample Chapter: ATtiny I/O Ports

By Warwick A. Smith (South Africa)



I/O ports control the pins of a microcontroller and allow them to be individually configured as input pins or output pins. That's such a broad statement, it should fit just about any newbie course or introduction to microcontroller programming. However, to truly comprehend and exploit a microcontroller's I/O abilities, you need to delve deeper into the machine. In this article, Elektor book author Warwick Smith pulls it off by demo-ing some assembly-code programming on the popular ATtiny micro. Convincing? Let's check it out!

Editor's Note. *This article is an excerpt from the 376-page book* Explore ATtiny Microcontrollers using C and Assembly Language (W. Smith, Elektor, 2021) *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.*

I/O ports are configured and controlled using a set of four registers in the ATtiny13(A) and ATtiny25/45/85. When a pin is configured as an input pin, a program running on the AVR can read the logic level on the pin. If a switch is attached to the pin, the logic level on the pin can be read to see if the switch is open or closed. When a pin is configured as an output pin, it can be used to switch the logic level of the pin high (to logic level 1) or low (to logic level 0). An output pin can be used to drive an LED as was done in the LED blink project described elsewhere in the book.

Configuring I/O Pins as Outputs in Assembler

In this section, we look at how to configure more than one pin as an output using an 8-pin ATtiny with five LEDs attached to pins with series resistors. Code is used to configure the LEDs as a 5-bit binary counter that counts up from zero.

Figure 1 shows a circuit diagram of an ATtiny13(A) or ATtiny25/45/85 AVR microcontroller with five LEDs attached to I/O pins PB0 to PB4. Pin PB5 of the ATtiny microcontroller in the circuit is used as the debugWIRE pin for programming and debugging the microcontroller. Remember that to use this configuration a programmer/debugger, such as an Atmel-ICE, or AVR Dragon must be used.

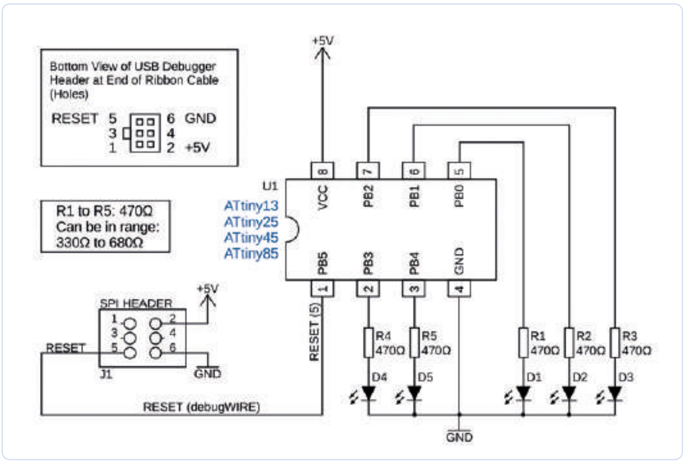


Figure 1: Five-LED Counter circuit diagram.

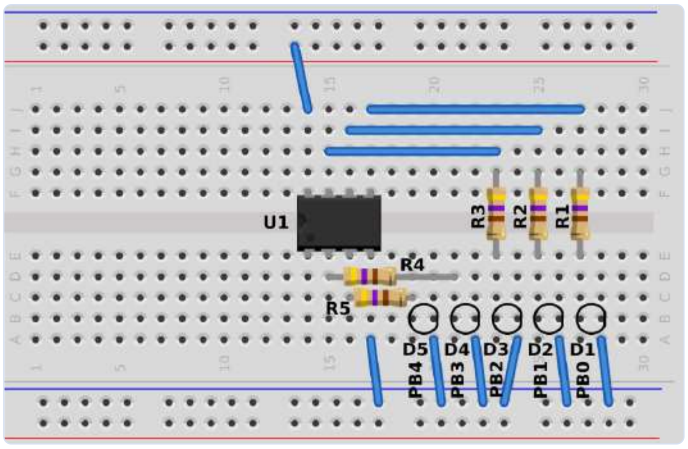


Figure 2: Five-LED count circuit breadboard layout.

A program-only USB programmer will not work in debugWIRE mode, and can not perform debugging. Reminder: **Do not set the `DWEN` fuse with a program-only USB programmer because it will not be able to take the AVR back out of debugWIRE mode**. The idea of using debugWIRE mode with this example circuit is to free up the other pins of the AVR that are normally used in ISP/SPI programming mode.

Program-Only Programmers
USB programmers with program-only capabilities can be used with the circuit of Figure 1 to load the example program that follows and see the count value displayed on the LEDs.

Wire the LEDs as shown in Figure 1 and **Figure 2**. The original USBasp design and the original USBtinyISP design both have protection resistors on the lines that drive out from these programmers, protecting the programmer, and the target AVR chip. The Arduino Uno programmed as an ArduinoISP does not have any protection resistors, but these can be added on the Arduino Uno MOSI line and SCK line which drive into the target AVR. 270-Ω protection resistors are used in the original USBasp design, and are placed on the MOSI, SCK and RESET lines. 1k5 resistors are used on the MOSI and RESET lines of the original USBtinyISP design.

Protection resistors prevent a short-circuit should one of the target AVR pins drive an output voltage at the same time that the programmer drives an output voltage of opposite polarity.

Peripheral Hardware Devices Interfering with Programming
Although the circuit of Figure 1 can be programmed using the ISP/SPI interface with the LEDs and series resistors attached, other circuits may have hardware attached that interferes with programming. If any peripheral hardware attached to pins of an AVR to be programmed will interfere with programming the AVR, there are some solutions to this problem. Of course, those readers who have a debugWIRE capable USB programmer/debugger, such as the Atmel-ICE, can simply put the target AVR into debugWIRE mode, and thereby use only one pin for programming. Another solution is to program the AVR on another breadboard using ISP/SPI, and then plug it into the target circuit afterwards.

Alternatively, an AVR with more pins can be used, but the free port pins don't always match the same pins from the same port of an 8-pin PDIP ATtiny. For example, if using port pins PB0 to PB4 like the circuit of Figure 1 does, there are not five consecutive free pins available on the 14-pin ATtiny24/44/84 range of AVRs, because the ISP/SPI pins use up pins from both port A and port B. On the 20-pin ATtiny26/261/461/861 range, the whole of port A is free with an ISP/SPI programmer connected, but this means that the software needs to be changed to use port A instead of port B. Fortunately, the 20-pin ATtiny2313/4313 range does have pins PB0 to PB4 free with an ISP/SPI programmer connected.

Putting the AVR into debugWIRE Mode
In order for the microcontroller to be programmed using a single debugWIRE line as shown in Figure 1, it is necessary to first connect all of the ISP header lines to the microcontroller from the USB programmer/debugger and then set the `DWEN` fuse of the AVR to put it into debugWIRE mode. Your AVR will now be in debugWIRE mode. If not, attach your programmer/debugger, such as an Atmel-ICE or AVR Dragon, and ensure you are ready and able to set the `DWEN` fuse. Once the `DWEN` fuse is programmed, all of the ISP header connections can be removed from the circuit except RESET, +5 V (Vcc) and GND as Figure 1 shows.

Build the 5-LED ATtiny Circuit on Breadboard

If you have the hardware, then build the circuit of Figure 1 on an electronic breadboard. Make sure that the five LEDs are connected in a row on the breadboard with pins PB0 to PB4 connected in order starting at PB0 on the right. LED D1 will then be on the right of the row and D5 on the left. In other words, we want a single row of LEDs with PB0 connected to the LED on the right, PB1 connected to the LED next to it on the left, PB2 connected to the LED third from the right, and so on, as can be seen in the breadboard layout of Figure 2. The image shows just the outlines of the LEDs so that they do not block the view of the wire connections and resistors. If you do not have the hardware, then follow the programs using a simulator.

Assembly Code for the 5-LED Count Circuit

Start a new Microchip Studio AVR Assembler Project called *led_count_asm*. Type the code shown in **Listing 1** into *main.asm* of the project, replacing the skeleton code.

If you are using the hardware of Figure 1, then select the hardware tool (your debugger), such as the Atmel-ICE within Microchip Studio, with debugWIRE as the interface. You do so by clicking the *hammer* icon on the second top toolbar. If you are using the simulator, select *Simulator* as the tool instead. If using a "hobby" programmer, read on to be prompted to load the program to the target ATtiny.

The *led_count_asm* program configures pins PB0 to PB4 as output pins so that the LEDs attached to these pins can be driven on and off by the program. The program displays an incrementing binary number, or count, on the LEDs starting at 0 (represented by all LEDs off). When the count reaches its maximum value (all LEDs on) it wraps around to zero and starts counting up again. Each "off" LED represents a binary zero digit or logic 0 level, and each "on" LED represents a binary one digit, or logic 1 level. It is important to lay out the LEDs as shown in Figure 2 so that the count displays correctly with PB0/D1 as the LSB and PB4/D5 as the MSB of the count value.

Build the program and load it to the AVR if you are using the physical hardware of Figure 1 and Figure 2. If using an Atmel-ICE or AVR Dragon, use the *Start Without Debugging* icon on the top toolbar of Microchip Studio or keyboard shortcut *Ctrl + Alt + F5* to load the program to the AVR. If the debugger interface is set up correctly and the chip is in debugWIRE mode, the program will load and start running. The incrementing binary count will be seen on the LEDs. If using a program-only USB programmer such as a hobby programmer, then load the program to the target AVR using the appropriate function. If the program was typed correctly, the circuit wired correctly, and the program was saved and built after typing it in, you will see the binary count value counting up on the LEDs and you can use the simulator with the rest of the text that follows.

If you don't have the physical hardware, you can still see the count value by using the Simulator in Microchip Studio. The program can be stepped through by using the *Start Debugging* and *Break* icons. With the simulator started, open the I/O window from the top menu by clicking *Debug Windows I/O*, and then click on the *I/O Port (PORTB)* item. Step over the program using the *Step Over* icon or *F10* keyboard key. Look at the PORTB item at the bottom of the I/O window to see the count value that would be displayed on the LEDs if they were attached. The count is updated in PORTB each time that the `OUT` instruction is stepped over in the main loop.

How the LED Count Assembler Program Works

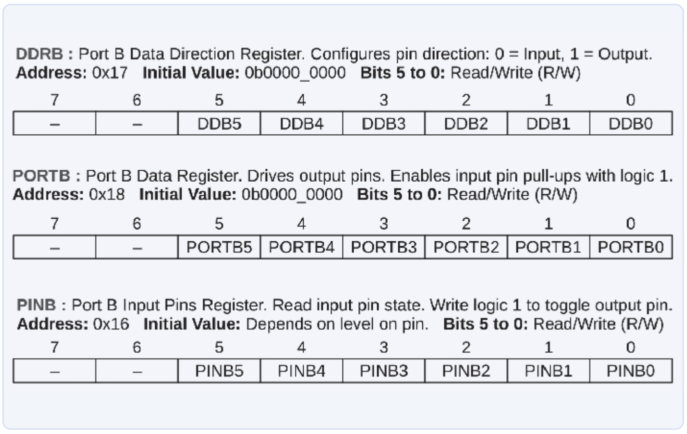
Half of the *led_count_asm* program code consists of the delay subroutine that was used in the *LED blink* program discussed elsewhere in the book. This subroutine is called once in the main loop of the program so that the count on the LEDs is visible to the eye without flashing past too fast. Have the *led_count_asm* project open in Microchip Studio, with the code from *main.asm* also open while following the explanation of the code that follows.

The first two instructions of the program are used to set pins PB0 to PB4 as output pins to drive the LEDs by setting bits in the DDRB register. **Figure 3** shows the DDRB register at the top of the figure. Each bit in this register corresponds to a pin on the microcontroller. For example, bit DDB0 corresponds to pin PB0, DDB1 corresponds to pin PB1, and so on.

When a bit in DDRB is set to logic 1, the corresponding pin becomes an output pin. If a bit in DDRB is cleared to logic 0, the corresponding pin becomes an input pin which is the default of all pins at power-up or reset. At the top of the program, 0b0001_1111 is first written to register R16, and then written from R16 to the DDRB register using the OUT instruction. It is necessary to first load the constant value to R16 because there is no instruction to directly write a constant value to an I/O register such as DDRB. Writing 0b0001_1111 to DDRB sets bits DDB0 to DDB4 making pins

Listing 1: led_count_asm : main.asm

```
    ; Set up pins PB0 to PB4 as output pins
     ldi     r16, 0b0001_1111
     out     DDRB, r16
     clr     r18              ; Clear count register
loop:
     out     PORTB, r18       ; Display count on LEDs
     rcall   delay
     inc     r18              ; Increment count
     andi    r18, 0b0001_1111 ; Clear unused bits
     rjmp    loop
; Delay subroutine
delay:
     ldi     r16, 0xff
delloop1:
     ldi     r17, 0xff
delloop2:
     dec     r17
     brb     SREG_Z, delloop2
     dec     r16
     brbc    SREG_Z, delloop1
     ret
```



DDRB : Port B Data Direction Register. Configures pin direction: 0 = Input, 1 = Output.
Address: 0x17 **Initial Value:** 0b0000_0000 **Bits 5 to 0:** Read/Write (R/W)

7	6	5	4	3	2	1	0
–	–	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0

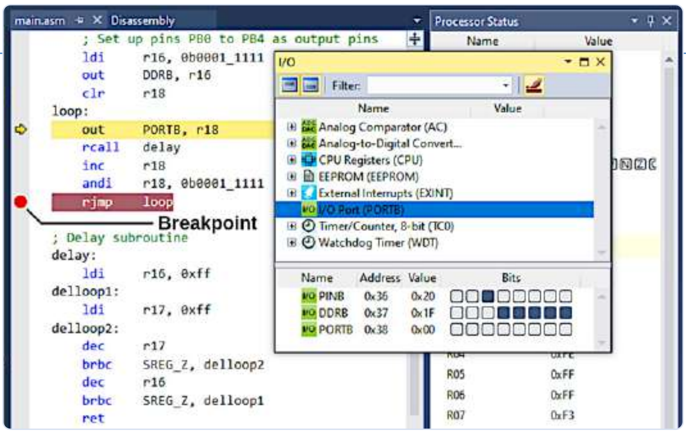
PORTB : Port B Data Register. Drives output pins. Enables input pin pull-ups with logic 1.
Address: 0x18 **Initial Value:** 0b0000_0000 **Bits 5 to 0:** Read/Write (R/W)

7	6	5	4	3	2	1	0
–	–	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0

PINB : Port B Input Pins Register. Read input pin state. Write logic 1 to toggle output pin.
Address: 0x16 **Initial Value:** Depends on level on pin. **Bits 5 to 0:** Read/Write (R/W)

7	6	5	4	3	2	1	0
–	–	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0

Figure 3: ATtiny13(A) and ATtiny25/45/85 I/O port registers.



```
    ; Set up pins PB0 to PB4 as output pins
    ldi     r16, 0b0001_1111
    out     DDRB, r16
    clr     r18
loop:
    out     PORTB, r18
    rcall   delay
    inc     r18
    andi    r18, 0b0001_1111
    rjmp    loop
; Delay subroutine
delay:
    ldi     r16, 0xff
delloop1:
    ldi     r17, 0xff
delloop2:
    dec     r17
    brbc    SREG_Z, delloop2
    dec     r16
    brbc    SREG_Z, delloop1
    ret
```

Name	Address	Value	Bits
PINB	0x36	0x20	☐☐■☐☐☐☐☐
DDRB	0x37	0x1F	☐☐☐■■■■■
PORTB	0x38	0x00	☐☐☐☐☐☐☐☐

Name	Value
R05	0xFF
R06	0xFF
R07	0xF3

Figure 4: Inserting a Breakpoint in the Microchip Studio Debugger.

PB0 to PB4 output pins.

Although there are four registers for controlling I/O port B, only three are shown in Figure 3. The fourth register, MCUCR has only one bit that applies to port B. This bit is a global pull-up disable bit that we do not use in this chapter. Refer to the Register Description part of the datasheet in the I/O Ports section to see this register — use either the datasheet for the ATtiny13, ATtiny13A, or ATtiny25/45/85.

R18 is used in the program to hold an incrementing count value that is displayed on the LEDs. R18 is cleared to 0 using `CLR` before the main loop so that the count starts from 0.

In the main loop, the contents of R18 are sent to the PORTB register using the OUT instruction. The PORTB register can be seen in the middle of Figure 3. Again, each bit in this register corresponds to a pin on the microcontroller. For pins that were set up as output pins using DDRB, the logic levels written to the PORTB register appear on these pins. For pins that are set up as input pins, a logic 1 in PORTB enables an internal pull-up resistor on these pins. A logic 0 disables the pull-up resistor on the corresponding pin.

Because pins PB0 to PB4 are set up as output pins, the count value written to PORTB appears on these pins as logic levels that switch the LEDs on and off. A logic 1 in the count value switches the corresponding LED on, and a logic 0 level in the count value switches the corresponding LED off.

After the count value is written to PORTB using the `OUT` instruction, the delay subroutine is called to leave the count value on the LEDs for a while. Register R18 is then incremented by 1 using the `INC` instruction so that the count value that it is holding increments. `ANDI` is used to clear the top three bits of the count value in R18, by using a mask value of 0b0001_1111. This ensures that these top bits are always zero. Back at the top of the loop when R18 is written to PORTB again, 0 is always written to the top three bits, because they were cleared using `ANDI`. When the end of the loop is reached at the `RJMP` instruction, program execution starts at the top of the loop again, with the new count value written to PORTB, and displayed on the LEDs.

Some things to note about this program: I/O registers DDRB and PORTB are both used to set up and control port B of the microcontroller in both programs. The LED blink program only needs to control a single pin, so uses the `SBI` and `CBI` instructions to set and clear a single bit in the I/O registers directly — none of the working registers R0 to R31 are needed for this. The LED count program needs to access five LEDs or pins at the same time so uses the `OUT` instruction to write to multiple bits in a register at the same time.

Register usage is important to keep track of in an assembly language program. For example, an immediate value is loaded to R16 at the beginning of the program. R16 is only used temporarily in this instance, so can be used later in the program without the need to first save its value. It is used again in the delay subroutine. Because R16 and R17 are used in the delay subroutine, R18 was chosen to hold the count value and is not used for anything else. This ensures that the count value is never overwritten. If a program gets very big and there are no spare registers to use for dedicated purposes, then `PUSH` and `POP` instructions can be used at the start and end of a subroutine to preserve the values in registers that are used.

Using a Breakpoint in the Debugger

With the simulator, or the physical hardware and a hardware debugger such as the Atmel-ICE, use the debugger to step through the program by clicking *Start Debugging* and *Break* as we have done before. View the I/O port registers by opening the I/O window in Microchip Studio. With the debugger running, select *Debug Windows I/O* and then click *I/O Port (PORTB)* in the I/O window as shown in **Figure 4**.

When program execution enters the main loop, it is convenient to place a breakpoint on the `RJMP` instruction and execute the entire loop by clicking the *Continue* toolbar button (keyboard key *F5*). A breakpoint can be placed on the `RJMP` instruction by clicking the gray area at the very left of the instruction and of the Microchip Studio window. This places a red dot in this area indicating that a breakpoint has been set on the instruction, as can be seen in Figure 4. Alternatively, click the instruction that you want to place a breakpoint on so that the cursor is placed on it, and then select *Debug Toggle Breakpoint* from the top menu, or use the *F9* keyboard key. Once the breakpoint is set, use the *Continue* toolbar button or press *F5* to step though the entire loop and only break at the bottom. In this way, the count can be seen incrementing by 1 in the PORTB in the *I/O window* as well as in register R18 in the *Processor Status window*, without having to individually step over each instruction in the loop.

Remove the breakpoint from the `RJMP` instruction by clicking the red dot at the left of the instruction, or use the menu or *F9* to toggle the breakpoint off, assuming that the cursor is on the `RJMP` instruction line. Put a breakpoint on the `INC R18` instruction. Now use *F5* to step through the loop. The new breakpoint ensures that the PORTB register and R18 contain the same value when program execution stops. If program execution stops on `RJMP`, then R18 is incremented before writing its new value to PORTB, thus they are unsynchronized when observing these values in the debugger windows in Microchip Studio.

The software released by the author in support of the book is available for free downloading. Head over to [1], scroll down to the *Downloads*, and click on the file name: *Software_Explore ATtiny Microcontrollers using C and Assembly Language*. Save the ZIP archive file locally (approx. 29 kB) and then extract it.

220045-01

Contributors

Text and Graphics: **Warwick Smith**
Editor: **Jan Buiting**
Layout: **Giel Dols**

Questions or Comments?

Do you have any technical questions or comments related to this article? Email the author at warwsmi@axxess.co.za or Elektor at editor@elektor.com.

WEB LINK

[1] Book resources/info page: www.elektor.com/20007

The WinUI Graphics Framework for Windows Apps

A Small Demo Application

By **Dr. Veikko Krypczyk** (Germany)

Software designed to control various electronic applications often runs in a Windows environment. Using a locally installed desktop application gives you direct access to all of the PC's system interfaces. Microsoft is currently rationalizing its support for developers working with Windows interface frameworks. The new graphics interface WinUI 3 is the way forward. Here, we will take a look at the technical background and show how to use it by building a small demo app for electronics technicians.

For many electronics projects, a Windows PC is used to provide control functions, data logging and other tasks; it will typically be necessary to develop a graphical user interface to run under Windows to carry out this function. Locally executed desktop applications are often the method of choice here; they give you complete access to the system environment and, via the appropriate drivers, also provide links to connected peripherals.

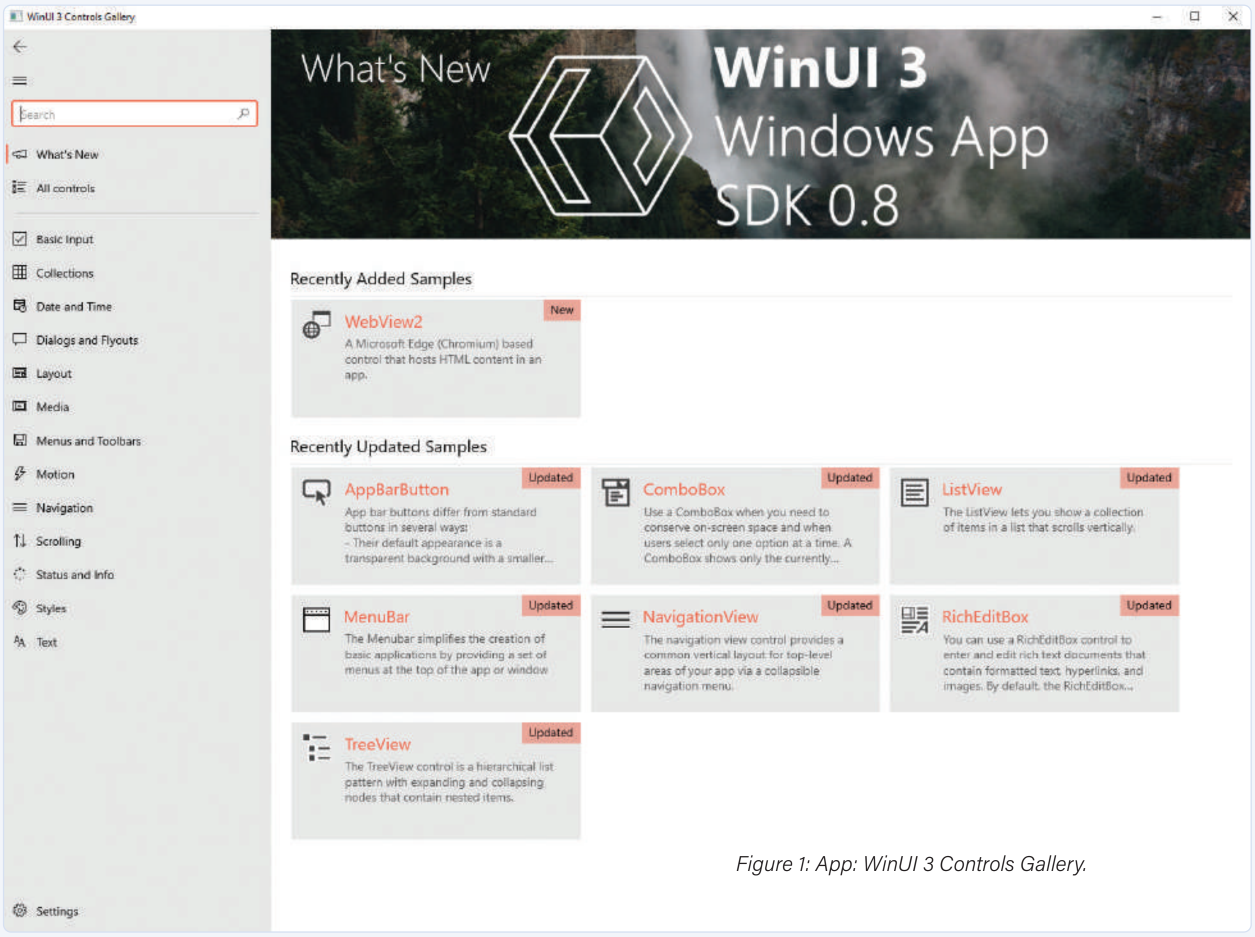
Microsoft, as the manufacturer of the Windows operating system, is currently undergoing a major technological upheaval. The focus of this activity is on the connectivity and design of user interfaces. The developments are coordinated under the project names *WinUI 3* and *Windows App SDK*. In this article, we show what this new graphic interface is all about and where we can use it. First, we get an overview of the possibilities of programming Windows applications with a graphical user interface. The purpose of WinUI 3 will become clear by using different technologies and application types. Not content with a purely theoretical review of this relatively new type of Windows application, we will also go ahead and create our first practical application.

The Technology

Basically, current applications using a graphical user interface for the Windows operating system can broadly be divided into two types. On the one hand, we have *Desktop Applications*. These are essentially based on the use of the *Win32 API*. There are different approaches, frameworks and programming languages for their development. The technologies Windows Forms (WinForms) and Windows Presentation Foundation (WPF) come from Microsoft. WinForms relies on the Windows GDI interface. WPF is based internally on DirectX and was originally intended as a replacement for WinForms. Both graphics frameworks were based on the .NET framework, which was intended for Windows programs and whose further development ended at version 4.8.

Version .NET 5, on the other hand, is the technological successor to .NET Core. This framework is not limited to Windows, but can also be used with other operating systems. Microsoft has surprisingly transformed both WinForms and WPF to .NET Core. If you are creating an app for Windows today and opt for WinForms or WPF, then you have the choice between the previous .NET framework and .NET Core. It is also possible to migrate existing applications, but, as is so often the case with such projects, it is often associated with a number of issues. Other manufacturers of development tools for Windows applications have mostly based them on the graphic interface of the operating system (GDI) and encapsulated this in their own framework.

The second category of Windows applications are apps for the *Universal Windows Platform (UWP)*. These run in an individually isolated area of the operating system and have only limited system access. Users install these apps through the store. In practice, however, this type of app has not proved to be popular and its uptake is quite low. One of the reasons for this is that system access from these apps is very limited. The UWP however has the advantage that the graphics framework WinUI 2 used here is significantly more modern than the technologies of WinForms and WPF. An appealing design, new visual components, the use of materials and the orientation towards the design language *Fluent Design System* are its stand out features. In other words, apps for the UWP look modern, contemporary and fresh, but their usability is somewhat limited. To achieve similar effects with the WinForms or WPF technologies, we need to do some pull-ups, use extensive third-



Recently Added Samples

WebView2 — New
A Microsoft Edge (Chromium) based control that hosts HTML content in an app.

Recently Updated Samples

AppBarButton — Updated
App bar buttons differ from standard buttons in several ways:
- Their default appearance is a transparent background with a smaller...

ComboBox — Updated
Use a ComboBox when you need to conserve on-screen space and when users select only one option at a time. A ComboBox shows only the currently...

ListView — Updated
The ListView lets you show a collection of items in a list that scrolls vertically.

MenuBar — Updated
The Menubar simplifies the creation of basic applications by providing a set of menus at the top of the app or window

NavigationView — Updated
The navigation view control provides a common vertical layout for top-level areas of your app via a collapsible navigation menu.

RichEditBox — Updated
You can use a RichEditBox control to enter and edit rich text documents that contain formatted text, hyperlinks, and images. By default, the RichEditBox...

TreeView — Updated
The TreeView control is a hierarchical list pattern with expanding and collapsing nodes that contain nested items.

Figure 1: App: WinUI 3 Controls Gallery.

party components or "mix" the WinForms/WPF technologies with the UWP. This approach however quickly leads to a more complex app structure and brings with it the typical disadvantages, such as higher susceptibility to errors and poorer maintainability.

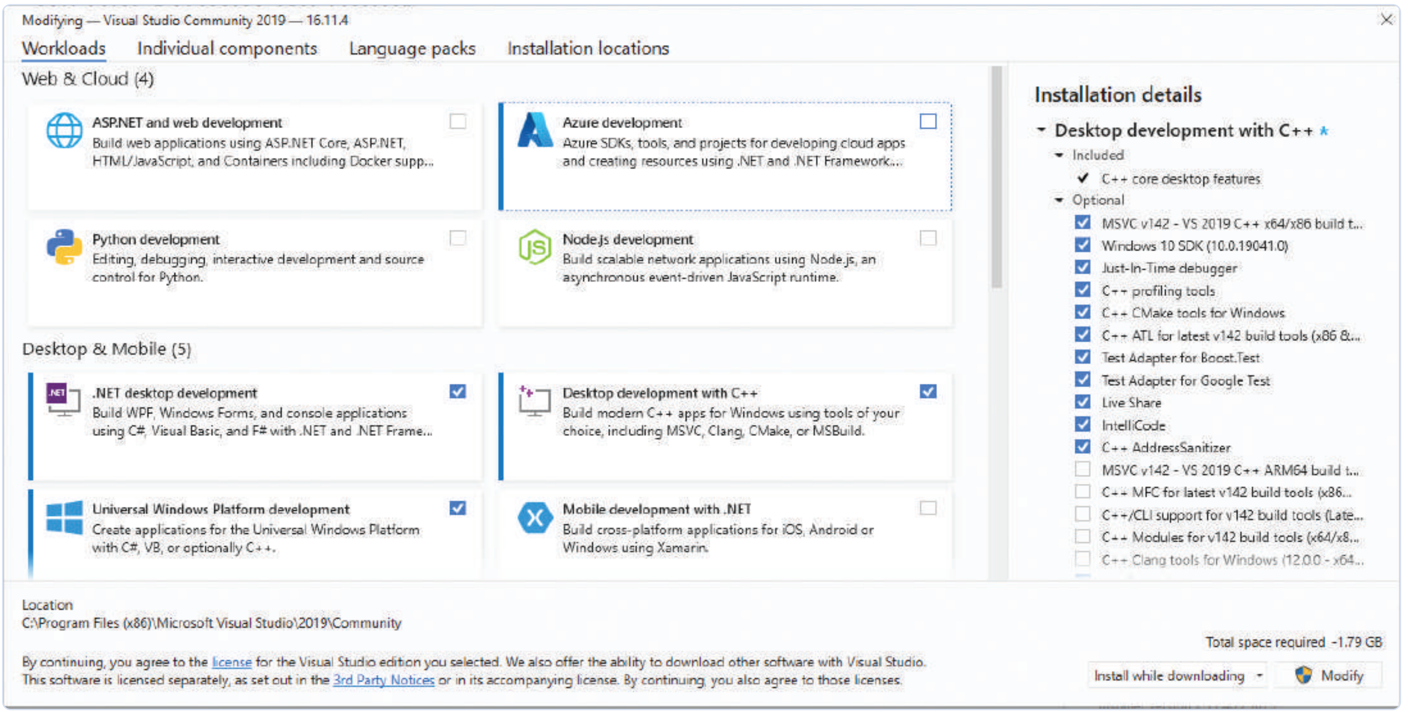
Software designed for electronic control, development, etc., are almost without exception classic desktop applications. You can also build these using other tools and frameworks. For the programming language Java there is, for example, the graphics framework *Swing*, which for Windows is based internally on the GDI operating system interface.

The WinUI 3 Graphic Framework

With the introduction of the WinUI 3 graphic interface, Microsoft would like to enable all applications under Windows to use a modern graphic interface. WinUI 3 is the technological successor to WinUI 2 [1]. However, it is available for all types of Windows applications and is therefore not limited to use in apps for the UWP. WinUI 3 is part of the new Windows App SDK, which is also provided in parallel with the introduction of Windows 11. The Windows APP SDK bundles new features for the development of Windows applications. It is not only aimed at Windows 11, but can also be used under the current versions of Windows 10. Development of the Windows App SDK is still ongoing, but a first version is available that can already be used in newly created applications.

WinUI 3 is technically and conceptually based on WinUI 2. If you have ever developed an app for the UWP, you will quickly get to grips with it. It is based on the following principles:

- *Separation of code and design:* The user interface is declaratively created in separate files using the XML-based XAML language.
- *UI control elements*: A range of controls are available for designing the user interface. These include basic elements such as buttons and text entry fields, along with more complex and advanced elements such as a calendar control element, a WebView or an element for displaying personal data, which we can use, for example, for user administration. If you would like to explore the range of control elements further go to the Microsoft Store and download the *WinUI 3 Controls Gallery* app from there. This app gives a preview of the available controls for WinUI 3. Their use (Use Case) and their integration in the source code (XAML) are demonstrated. Corresponding links to the documentation can also be found (**Figure 1**).
- *Loose coupling through DataBinding*: The properties and events of the control elements are linked to the source code by means of data binding. In this way, data is exchanged in both directions between the program code and the user interface control. Events of the control elements, such as the click on a button, are also forwarded to the relevant algorithm in the same way.
- *Modern Design*: The WinUI 3 provides a contemporary feel. This includes the use of Microsoft's design language *Fluent Design System* with the *Mica* material introduced in Windows 11. The Fluent Design System provides the following UI elements: conscious use of geometry and colour, overlapping of surfaces, use of selected materials and the use of specific iconography and typography for visual design using images, symbols and fonts. Motions between UI elements are also supported.


Modifying — Visual Studio Community 2019 — 16.11.4

Web & Cloud (4)

- ASP.NET and web development — Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker supp...
- Azure development — Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET and .NET Framework...
- Python development — Editing, debugging, interactive development and source control for Python.
- Node.js development — Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Desktop & Mobile (5)

- ✓ .NET desktop development — Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...
- ✓ Desktop development with C++ — Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.
- ✓ Universal Windows Platform development — Create applications for the Universal Windows Platform with C#, VB, or optionally C++.
- Mobile development with .NET — Build cross-platform applications for iOS, Android or Windows using Xamarin.

Installation details

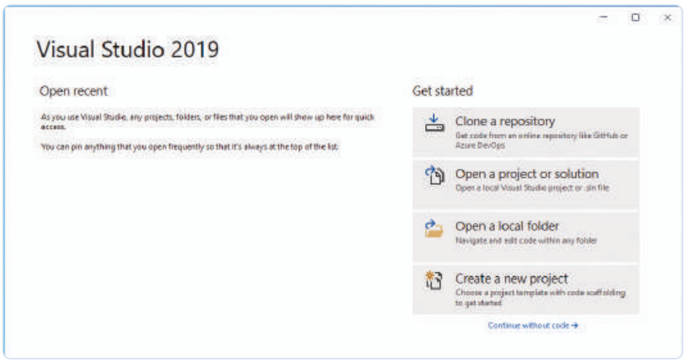
- Desktop development with C++ *
  - Included
    - ✓ C++ core desktop features
  - Optional
    - ✓ MSVC v142 - VS 2019 C++ x64/x86 build t...
    - ✓ Windows 10 SDK (10.0.19041.0)
    - ✓ Just-In-Time debugger
    - ✓ C++ profiling tools
    - ✓ C++ CMake tools for Windows
    - ✓ C++ ATL for latest v142 build tools (x86 &...
    - ✓ Test Adapter for Boost.Test
    - ✓ Test Adapter for Google Test
    - ✓ Live Share
    - ✓ IntelliCode
    - ✓ C++ AddressSanitizer
    - MSVC v142 - VS 2019 C++ ARM64 build t...
    - C++ MFC for latest v142 build tools (x86...
    - C++/CLI support for v142 build tools (Late...
    - C++ Modules for v142 build tools (x64/x8...
    - C++ Clang tools for Windows (12.0.0 - x64...

Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required -1.79 GB

Figure 2: Select the necessary Workloads for Visual Studio.


Visual Studio 2019

Open recent

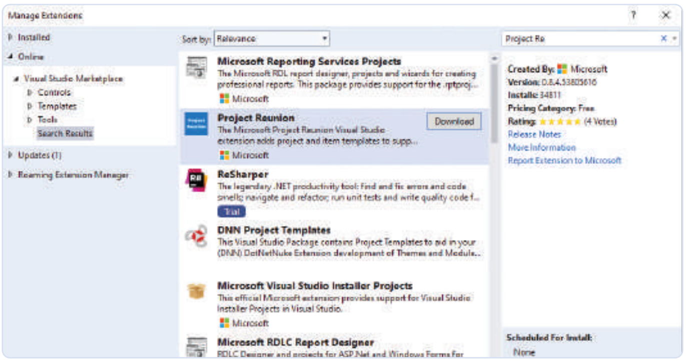
As you use Visual Studio, any projects, folders, or files that you open will show up here for quick access.

You can pin anything that you open frequently so that it's always at the top of the list.

Get started

- Clone a repository — Get code from an online repository like GitHub or Azure DevOps
- Open a project or solution — Open a local Visual Studio project or .sln file
- Open a local folder — Navigate and edit code within any folder
- Create a new project — Choose a project template with code scaffolding to get started

Figure 3: First Start of Visual Studio.


Manage Extensions

- Microsoft Reporting Services Projects — The Microsoft RDL report designer, projects and wizards for creating professional reports. This package provides support for the .rptproj... — Microsoft
- Project Reunion — The Microsoft Project Reunion Visual Studio extension adds project and item templates to supp... — Microsoft — Download
- ReSharper — The legendary .NET productivity tool: Find and fix errors and code smells; navigate and refactor; run unit tests and write quality code f... — Trial
- DNN Project Templates — This Visual Studio Package contains Project Templates to aid in your (DNN) DotNetNuke Extension development of Themes and Module...
- Microsoft Visual Studio Installer Projects — This official Microsoft extension provides support for Visual Studio Installer Projects in Visual Studio. — Microsoft
- Microsoft RDLC Report Designer — RDLC Designer and projects for ASP.Net and Windows Forms for

Created By: Microsoft
Version: 0.8.4.53805616
Installs: 34811
Pricing Category: Free
Rating: ★★★★★ (4 Votes)

Scheduled For Install:
None

Figure 4: Windows SDK App (Project "Reunion") installation.

Let's look at the procedure for developing applications with the WinUI 3. First, we will need to set up the development environment.

The Development Environment and Setup

The current version of Visual Studio 2019 [2] will be used as the development environment. The Community Edition will be sufficient for our needs. By the time you're reading this article, a stable version of Visual Studio 2022 may already be available, in which case you should use this version. It is advisable to install the latest updates for the operating system beforehand. During the installation of Visual Studio, you will be asked to select the installation packages. You can also call up the *Visual Studio Installer* at any point later on via the start menu. Now select the following installation packages (Workloads): *.NET desktop development, Desktop development with C ++* and *Universal Windows Platform development* (**Figure 2**). After installation, start Visual Studio and in the start screen select the option *Continue without code ->* (**Figure 3**).

We now also need to install the template for development with WinUI 3. To do this in Visual Studio, choose the menu option *Extensions | Manage Extensions*. Search here for *Project Reunion* (the development name of the new Windows App SDK) and install the current version (**Figure 4**). In the same way install the *Windows Template Studio* extension. This offers advanced templates for creating a new application. Visual Studio must be restarted after the extensions have been downloaded; the installation will then take place automatically.

An App for WinUI 3

Let's start by creating a new project. Here in **Figure 5** we select the App *WinUI 3 in Desktop* template (*Windows Template Studio*). In the Windows Template Studio (**Figure 6**), we can configure the project:

- *Project type*: Here we specify the type of navigation, for example with a menu bar or a side navigation bar (Hamburger menu).
- *Design pattern*: Direct installation and configuration of the MVVM toolkit. This couples the elements of the user interface (defined in XAML) with the program logic (programming language C#).
- *Pages*: We can add a number of pages to the project. We can choose from different templates, for example one page for entering program settings.
- *Features*: Here we can select some sample themes or save program settings.

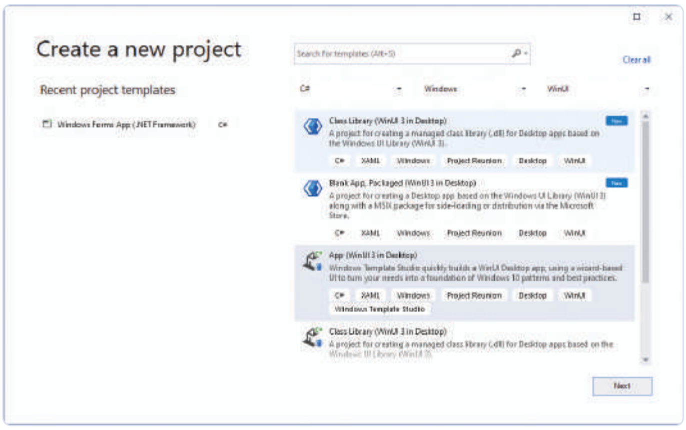
By clicking on the *Create* button, we generate the desktop application that WinUI 3 uses. Windows Template Studio generates a project folder with three projects:

- *App*: This contains the source code for the desktop application. In the subfolder *View*, for example, you will find the XAML files for the pages that were created by Windows Template Studio. The program logic is stored in files in the *ViewModel* program folder.
- *Package*: The project is responsible for providing the desktop application. At the moment, WinUI 3 applications are installed on the target computer as an app package. This format has so far been used for UWP apps. The generated packages can also be distributed by using Store. Future versions of the Windows App SDK should also allow installation without an app package.
- *Core*: This project contains the collection of services and classes that provide service for the app. This project is not mandatory and can be left out.

Start the application directly from Visual Studio using the green arrow on the toolbar. Congratulations - You have created your first application with WinUI 3 (**Figure 7**). I should emphasis again; this is a desktop application with full system access. As already mentioned, this is important for software designed to control external electronics, for example. A side navigation bar, first pages and the possibility to adapt the application design are all possible. You have all options for accessing the system, including communication with the system libraries and drivers. Now we can go ahead and experiment with the design of the user interface.

A Demo Application

The best way to become familiar with any new system is to try it out. Here we will design a simple user interface for our first application (the source code, for this example, can be found on the web page for the article [3]). The starting point is the XAML file for the relevant page. As an experiment we can create a handy calculation tool for use with the LM317 adjustable linear voltage regulator (**Figure 8**). The output voltage of this device is given by the formula $Vout = 1.25\ (1 + R2/R1)$. We can solve this equation for $R2$ and thus calculate its value to give the desired output voltage. With the help of this example we can


Create a new project

Recent project templates

Windows Forms App (.NET Framework) C#

Search for templates (Alt+S) | Clear all

C# | Windows | WinUI

Class Library (WinUI 3 in Desktop) — New
A project for creating a managed class library (.dll) for Desktop apps based on the Windows UI Library (WinUI 3).
C# XAML Windows Project Reunion Desktop WinUI

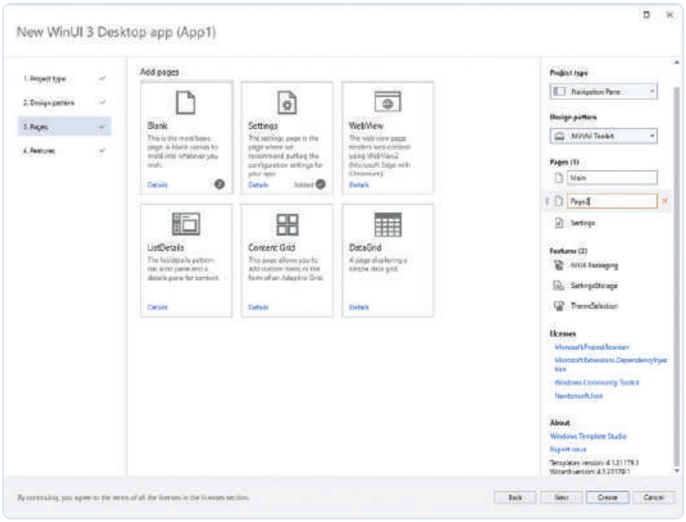
Blank App, Packaged (WinUI 3 in Desktop) — New
A project for creating a Desktop app based on the Windows UI Library (WinUI 3) along with a MSIX package for side-loading or distribution via the Microsoft Store.
C# XAML Windows Project Reunion Desktop WinUI

App (WinUI 3 in Desktop)
Windows Template Studio quickly builds a WinUI Desktop app, using a wizard-based UI to turn your needs into a foundation of Windows 10 patterns and best practices.
C# XAML Windows Project Reunion Desktop WinUI Windows Template Studio

Class Library (WinUI 3 in Desktop)
A project for creating a managed class library (.dll) for Desktop apps based on the Windows UI Library (WinUI 3).

Next

Figure 5: Project template WinUI 3 Desktop.


New WinUI 3 Desktop app (App1)

1. Project type
2. Design pattern
3. Pages
4. Features

Add pages

Blank — This is the most basic page. A blank canvas to mold into whatever you wish. Details

Settings — The settings page is the page where we recommend putting the configuration settings for your app. Details Added

WebView — The web view page renders web content using WebView2 (Microsoft Edge with Chromium). Details

ListDetails — The list/details pattern has a list pane and a details pane for content. Details

Content Grid — This page allows you to add custom items in the form of an Adaptive Grid. Details

DataGrid — A page displaying a simple data grid. Details

Project type: Navigation Pane
Design pattern: MVVM Toolkit
Pages (3): Main, Page1, Settings
Features (3): MSIX Packaging, SettingsStorage, ThemeSelection
Licenses: Microsoft.ProjectReunion, Microsoft.Extensions.DependencyInjection, Windows Community Toolkit, Newtonsoft.Json
About: Windows Template Studio, Report issue
Templates version: 4.1.21179.1
Wizard version: 4.1.21179.1

By continuing, you agree to the terms of all the licenses in the licenses section.

Back | Next | Create | Cancel

Figure 6: Windows Template Studio.

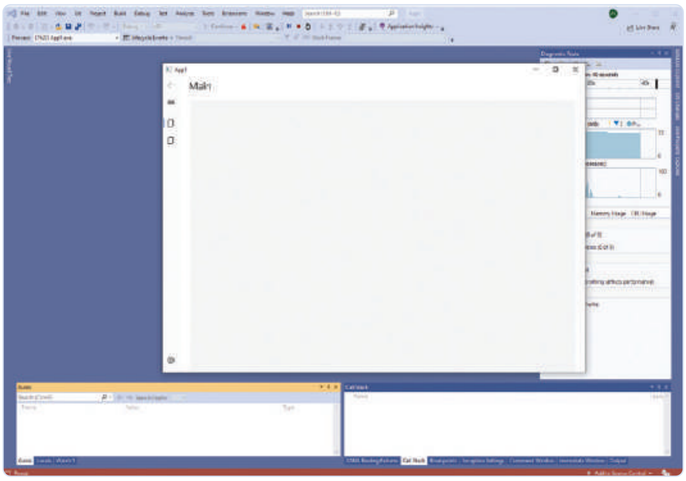


Figure 7: A first desktop application with WinUI 3.

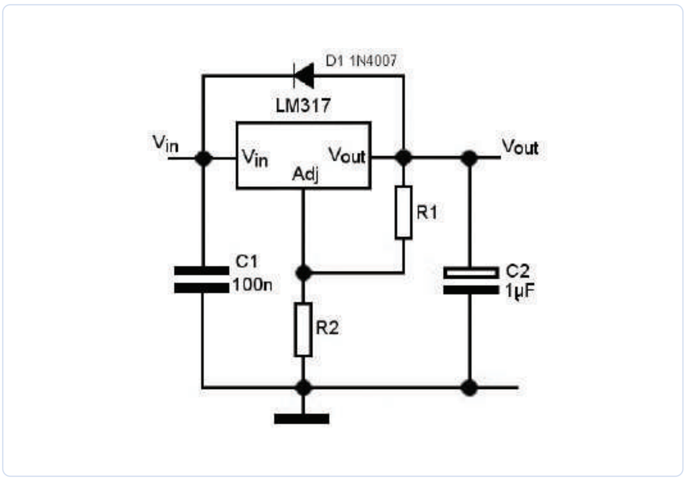


Figure 8: LM317 voltage regulator circuit diagram.

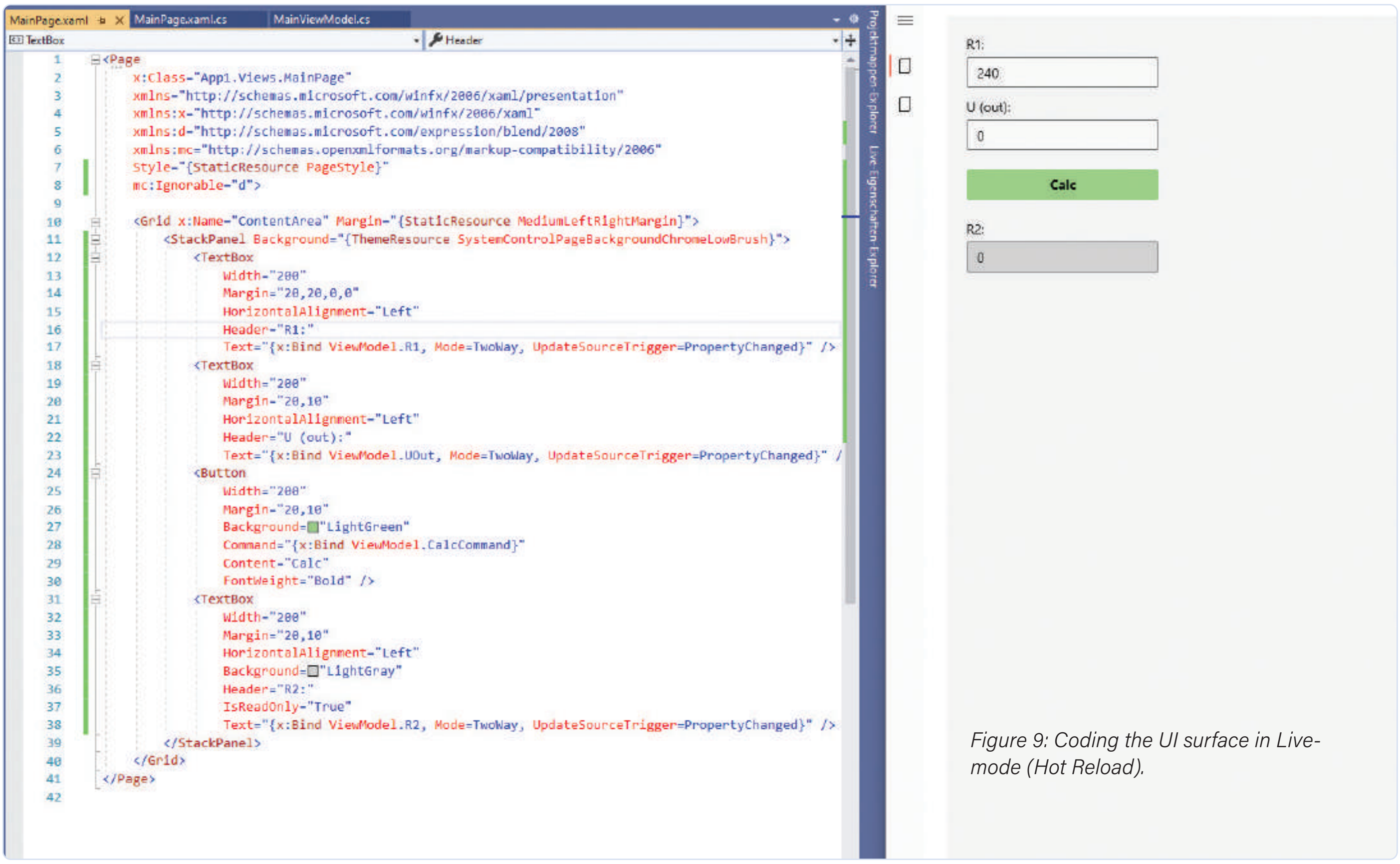
demonstrate the procedure for programming applications with WinUI 3. This process will include the following steps:

- Definition of the User Interface in XAML.
- Coding the Program logic in C#.
- Binding the user interface to the program logic.
- Forwarding the user interaction from the user interface to the program logic.
- Output the data to the form.

Let's start by defining the surface. We need two text fields to record the values of *R1* and V_{Out}. We also need a text field for the value of *R2*. A *Button* will be required to trigger the calculation. We use *TextBox* controls for input and output. All the elements should be arranged in the form of a vertical stack one above the other, which is why they are inserted into a layout container of the type *<StackPanel />*. Without

Table 1: Control Elements

Control Element	Property	Value	Description
TextBoxR1	Width	200	Textbox width
	Margin	20, 20, 0, 0	Margin size: left, top, right, bottom
	HorizontalAlignment	Left	Left justified
	Header	R1:	Caption
	Text	x:Bind ViewModel.R1	Binding the property to the variable R1 in C#.
TextBoxU (out)	Width	200	Textbox width
	Margin	20, 10	Margin size: left, top, right, bottom
	HorizontalAlignment	Left	Left justified
	Header	U (out):	Caption
	Text	x:Bind ViewModel. UOut	Binding the property to the variable UOut in C#.
TextBoxR2	Width	200	Textbox width
	Background	LightGray	Background colour
	Margin	20, 10	Margin size: left, top, right, bottom
	HorizontalAlignment	Left	Left justified
	Header	R2:	Caption
	IsReadOnly	True	Read only protection
	Text	x:Bind ViewModel.R2	Binding the property to the variable R2 in C#.
Button	Width	200	Button area width
	Margin	20, 10	Margin size: left, top, right, bottom
	Background	LightGreen	Background colour
	Command	x:Bind ViewModel. CalcCommand	Binding to the CalcCommand Method in C#.
	Content	Calc	Caption
	FontWeight	Bold	Font properties



```
<Page
    x:Class="App1.Views.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    Style="{StaticResource PageStyle}"
    mc:Ignorable="d">

    <Grid x:Name="ContentArea" Margin="{StaticResource MediumLeftRightMargin}">
        <StackPanel Background="{ThemeResource SystemControlPageBackgroundChromeLowBrush}">
            <TextBox
                Width="200"
                Margin="20,20,0,0"
                HorizontalAlignment="Left"
                Header="R1:"
                Text="{x:Bind ViewModel.R1, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" />
            <TextBox
                Width="200"
                Margin="20,10"
                HorizontalAlignment="Left"
                Header="U (out):"
                Text="{x:Bind ViewModel.UOut, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" /
            <Button
                Width="200"
                Margin="20,10"
                Background=🟩"LightGreen"
                Command="{x:Bind ViewModel.CalcCommand}"
                Content="Calc"
                FontWeight="Bold" />
            <TextBox
                Width="200"
                Margin="20,10"
                HorizontalAlignment="Left"
                Background=⬜"LightGray"
                Header="R2:"
                IsReadOnly="True"
                Text="{x:Bind ViewModel.R2, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" />
        </StackPanel>
    </Grid>
</Page>
```

R1: 240

U (out): 0

Calc

R2: 0

Figure 9: Coding the UI surface in Live-mode (Hot Reload).

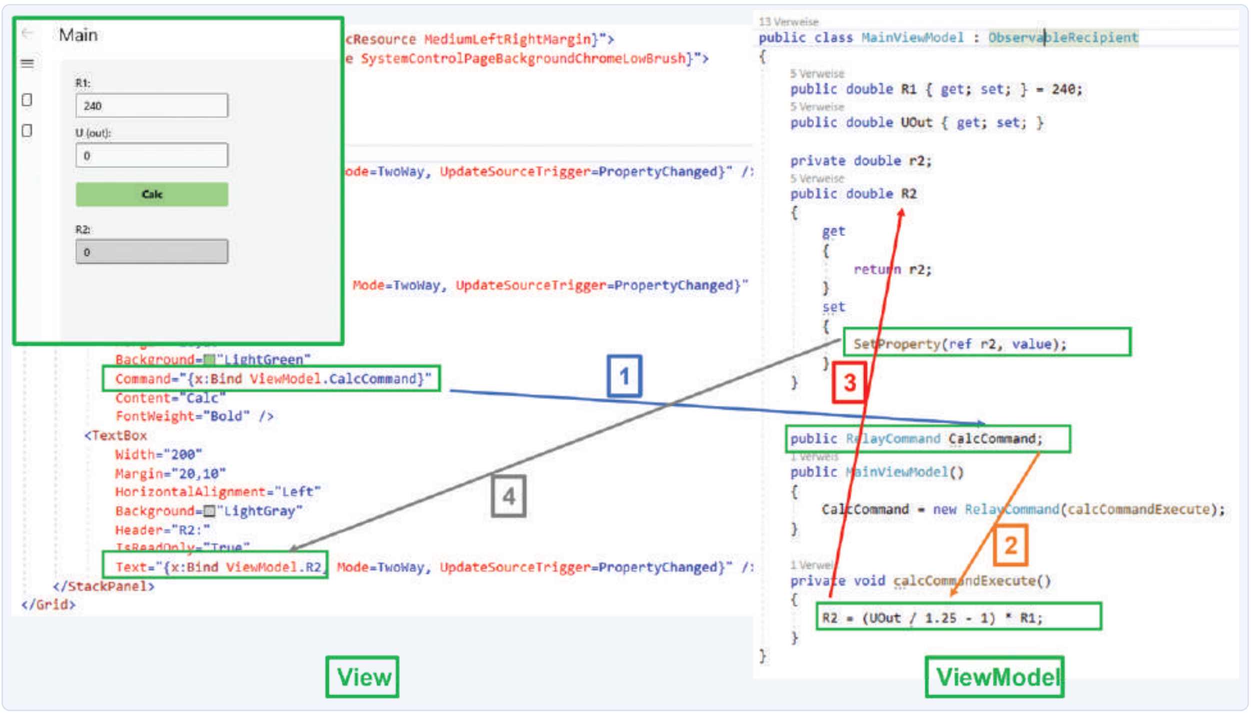
the need for any further configuration, all the elements will be arranged one above the other with *StackPanel*. The controls are configured via the XAML code, with the properties according to **Table 1**.

The associated source code is shown in **Listing 1**. You can code the surface interactively. Start the application and place the relevant XAML file in Visual Studio and the application side by side on the screen (**Figure 9**). Changes in the XAML code are immediately adopted when the application is started - without saving — and produce an updated display. This feature is called *Hot Reload* and is standard when creating graphical user interfaces.

What is interesting here is the control binding of the control elements of *TextBox* type with the properties of *Text*. Here you will find an expression according to the pattern in the XAML code:

Listing 1. Definition of User Interfaces

```
<Page
    x:Class="App1.Views.MainPage"
    …>

    <Grid x:Name="ContentArea" Margin="">
        <StackPanel Background="">
            <TextBox
                Width="200"
                Margin="20,20,0,0"
                HorizontalAlignment="Left"
                Header="R1:"
                Text="" />
            <TextBox
                Width="200"
                Margin="20,10"
                HorizontalAlignment="Left"
                Header="U (out):"
                Text="" />
            <Button
                Width="200"
                Margin="20,10"
                Background="LightGreen"
                Command=""
                Content="Calc"
                FontWeight="Bold" />
            <TextBox
                Width="200"
                Margin="20,10"
                HorizontalAlignment="Left"
                Background="LightGray"
                Header="R2:"
                IsReadOnly="True"
                Text="" />
        </StackPanel>
    </Grid>
</Page>
```



```
cResource MediumLeftRightMargin}">
e SystemControlPageBackgroundChromeLowBrush}">

ode=TwoWay, UpdateSourceTrigger=PropertyChanged}" /:

 Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"

        Background="LightGreen"
        Command="{x:Bind ViewModel.CalcCommand}"
        Content="Calc"
        FontWeight="Bold" />
    <TextBox
        Width="200"
        Margin="20,10"
        HorizontalAlignment="Left"
        Background="LightGray"
        Header="R2:"
        IsReadOnly="True"
        Text="{x:Bind ViewModel.R2, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" /:
  </StackPanel>
</Grid>
```

```
13 Verweise
public class MainViewModel : ObservableRecipient
{
    5 Verweise
    public double R1 { get; set; } = 240;
    5 Verweise
    public double UOut { get; set; }

    private double r2;
    5 Verweise
    public double R2
    {
        get
        {
            return r2;
        }
        set
        {
            SetProperty(ref r2, value);
        }
    }

    public RelayCommand CalcCommand;
    1 Verweis
    public MainViewModel()
    {
        CalcCommand = new RelayCommand(calcCommandExecute);
    }

    1 Verweis
    private void calcCommandExecute()
    {
        R2 = (UOut / 1.25 - 1) * R1;
    }
}
```

Figure 10: Example Relationship between View and ViewModel.

```
Text="{x:Bind ViewModel.R1, Mode=TwoWay, UpdateSource
   Trigger=PropertyChanged}"
```

This means that the property of *Text* is bound to the variable *R1*. This is defined in the *ViewModel* page and is based on the MVVM concept. UI events are handled in the *View* layer and data is managed in the *Model* layer. The ViewModel represents the connection between the two layers. Thanks to the MVVM concept, all layers are decoupled and can be developed and maintained independently of one another. Information on the MVVM pattern can be found under [4].

Program Logic

The program logic is implemented using C# (**Listing 2**). For this purpose, a program file (*ViewModel*) is assigned to each window of

Listing 2. Program logic for the calculation in C#

```
using CommunityToolkit.Mvvm.ComponentModel;
using CommunityToolkit.Mvvm.Input;

namespace App1.ViewModels
{
    public class MainViewModel : ObservableRecipient
    {
        public double R1 { get; set; } = 240;
        public double UOut { get; set; }

        private double r2;
        public double R2
        {
            get
            {
                return r2;
            }
            set
            {
                SetProperty(ref r2, value);
            }
        }


        public RelayCommand CalcCommand;
        public MainViewModel()
        {
            CalcCommand = new RelayCommand(calcComm
                andExecute);
        }

        private void calcCommandExecute()
        {
            R2 = (UOut / 1.25 - 1) * R1;
        }
    }
}
```

the user interface (*View*). In our example, it is the *MainViewModel* file assigned to the view *MainPage*. Here are some notes on the program code:

- **Import of libraries**: This is done via the *uses* statement. In our case we need two libraries for the MVVM pattern.
- **Definition of properties**: These must be *public* because we access the properties from outside, in this case from the view.
- **Automatic User Interface updating:** The `MainViewModel` class is derived from the `ObservableRecipient` base class, which was generated by the project wizard when the project was created. This class in turn implements the so-called `OnPropertyChanged` event. This ensures that when a value of a property changes, all bound elements are notified of the change. In our case the property `R2` is of interest. The value of `R2` is calculated in the program code. The so-called Setter of the property is called and the `OnPropertyChanged` event just described is triggered via the `SetProperty (…)` method. The *Text* property of the TextBox R2 is bound to the property `R2` (in the ViewModel). If `R2` is changed, the displayed value is automatically updated in the associated TextBox. This works thanks to data binding.
- **Pass on user actions by means of commands:** If the user presses the button, a command is triggered. The calculation method is linked to this command. Here, too, the user interface and program code are only linked to one another via the data link.
- **Assignment of View and ViewModel:** The program code (file: *MainViewModel.cs*) is assigned to the UI (file: *MainPage.xaml*). This is done in the page's code-behind file (file: *MainPage.xaml.cs*). You can see this if you take a look at the source code.
- **Calculation:** Calculation of value of `R2` takes place in the `calcCommandExecute (...)` method according to the above formula, which is then assigned to `R2`.

The user interface is therefore "loosely" linked to the program code by means of data binding. The connections to data binding just described are visualized using the example in **Figure 10**. Start the application and try it out. The value of the second resistor *R2* is calculated after entering the values of *R1* and V_{Out} (**Figure 11**).

We have thus described the basic development model for desktop applications with the WinUI 3 graphics framework. From today's perspective, it will become a new standard under Windows and can also be used by other development environments and languages. The variety of graphical options for creating modern applications is impressive.

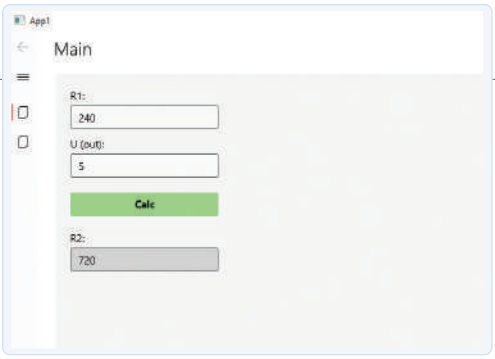


Figure 11: The finished app example.

Conclusions and Outlook

You can choose from a variety of technologies to create an application for the Windows operating system. The trend — also with a view to Windows 11 — is towards the use of WinUI 3. Using this will enable you to create an attractive and up to date user interface. From this perspective, it is worthwhile considering WinUI 3 when developing any new Windows application and to check migration options for any existing applications. The resulting applications have a contemporary interface and produce a good user experience. There are also no limitations on system access as there are with the UWP application model.

210407-01

Contributors

Text and images: **Dr. Veikko Krypczyk**
Editor: **Jens Nickel**
Translation: **Martin Cooke**
Layout: **Giel Dols**

Questions or Comments?

Do you have any technical questions or comments prompted by this article? If so, please contact the editor at editor@elektor.com.

WEB LINKS

[1] Information for WinUI 3: https://docs.microsoft.com/en-us/windows/apps/winui/
[2] Visual Studio 2019: https://visualstudio.microsoft.com/
[3] Project page for this article: http://www.elektormagazine.com/210407-01
[4] Information for MVVM pattern: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel

Off-Grid Solar Systems

Electrical Energy Independent of the Mains Grid

By Dr. Thomas Scherer (Germany)

What is an off-grid solar system? Where are such installations necessary or practical? What are the most important design considerations? These questions and more will be answered in this article.

In the September/October 2021 edition of *Elektor*, we took a look at a photovoltaic system connected to the mains grid [1]. Here we will consider essentially autonomous solar installations that are isolated from the public grid. These can be used to generate electrical energy where otherwise a grid connection would be too costly, such as in a shed on an allotment, or impossible, such as on a motorboat or sailing boat. In general these are low-power systems which can handle a peak demand in the range of a few watts to a couple of kilowatts. Also, as the levels of feed-in tariffs continue to fall, new and simplified designs for fixed domestic solar installations that store the generated energy locally in rechargeable batteries solely for private use, rather than feeding into the public supply grid, begin to make more sense. These installations typically have a maximum nominal power output of a few kWp ('kilowatts peak'). Let us now look at these small-scale systems in more detail.

Principle of Operation

An off-grid solar installation requires at least three components: the solar panel itself; some energy storage in the form of a rechargeable battery; and finally a charge controller that ensures that the battery is not overcharged. For smaller systems, typically operating at 12 V, that is in theory all that is needed. If, however, a 230 V AC output is required at 50 Hz or 60 Hz, a fourth component comes into play: an inverter. **Figure 1** shows a typical four-component solution: superficially it looks very simple, but as ever the devil is in the detail. In the following sections we will therefore take a look at these individual components.

Here's a real-life example: Klaus, a good friend of mine, decided to install a 12 V system in his shed because of the low prices and manageable size of the components involved. To design the system and specify the components there are two questions that first need to be answered.

Energy and Power

The first question to answer is how much total energy needs to be stored by the system. This directly affects the required capacity of the rechargeable battery and hence it is necessary to estimate the average load on the installation. Bearing on this calculation is the number of cloudy days that the system must be able to 'survive.' Klaus would like to use an electric drill in his shed, and brew the occasional cup of tea, but these relatively rare loads do not significantly affect the average load calculation. More significant is the requirement to keep cool beer always at hand: this calls for a 12 V refrigerator operating continuously with an average power draw of 20 W. The system should be able to run for at least one day without sun.

The second question is the peak power requirement. From this we can determine the maximum current that will be drawn from the rechargeable battery and hence also specify the parameters of the charge controller (and inverter, if used). Usually this question is very easy to answer: in the case of my friend's allotment shed the answer was 1 kW, covering the power draw of the water heater, a standard drill and possibly also a water pump, all operating at 230 V.

The Solar Panel

Over a 24-hour period the refrigerator in Klaus' shed will consume a maximum of 500 Wh. Although he lives in a sunny part of south-west Germany, the roof of his shed is unfortunately in the shadow of a tree and so the panel cannot be installed there. Instead it will have to be mounted vertically on the south-facing wall of the shed, which reduces its power output by some 30 % compared to mounting it at the optimal angle to the sun. The panel will therefore have to be over-specified by some 40 % to compensate for this loss. Fortunately there is plenty of space available and the price of panels has fallen considerably in recent years. An advantage of vertical mounting is that in winter snow will not lie on the panel, and moreover, since the sun is at a lower angle, the output will increase: in the best-case scenario the beer will be kept cool even on sunny winter days.

Now we can calculate the required power output from the panel. In this part of Germany we can reckon on an total incident energy of over 1200 kWh/m^2 over the course of a year. Over and above the expected daily energy use we should allow a safety margin of 100 %, and so for 500 Wh/day (from spring to autumn) we should be looking to generate 1 kWh/day. On the basis of 8 hours of sunshine per day we arrive at a required power output from the panel of around 125 Wp. On top of that we add the compensation for vertical mounting and arrive at 175 Wp. That means we need a 180 W panel, which will fit comfortably on the shed wall: see **Figure 2**.

The Rechargeable Battery or Batteries

The energy required to provide one day of reserve power is at least 500 Wh. At a nominal voltage of 12 V we will need a battery with a capacity of at least 40 Ah. Since our inverter is specified with an output power of 1 kW, we must also keep in mind that at maximum load it will be drawing a current of at least 85 A at its input. This is an important consideration in selecting a battery. First we must decide on the battery chemistry. A lithium battery pack rated at 40 Ah can comfortably deal with this current (about '2C,' or twice the current that the battery can deliver for one hour) because of its low internal resistance. However, such a pack can easily cost over €250/US$280/£210

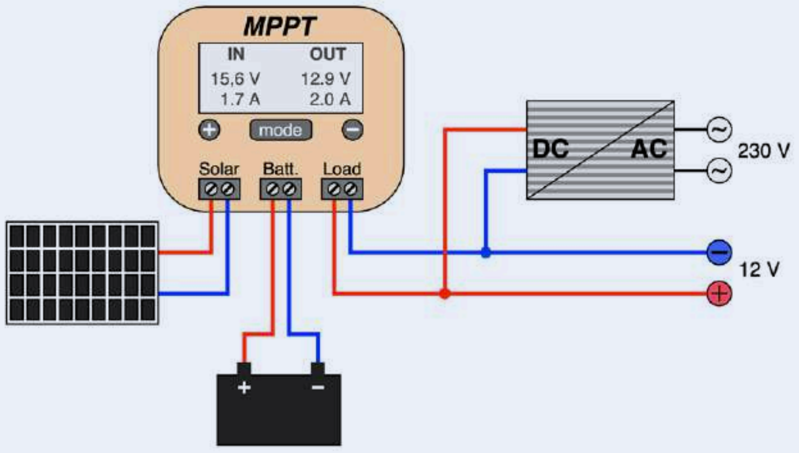


Figure 1: The standard wiring of the four components normally found in an off-grid solar installation. The inverter on the right is only required if it is desired to run equipment designed for 230 V operation.

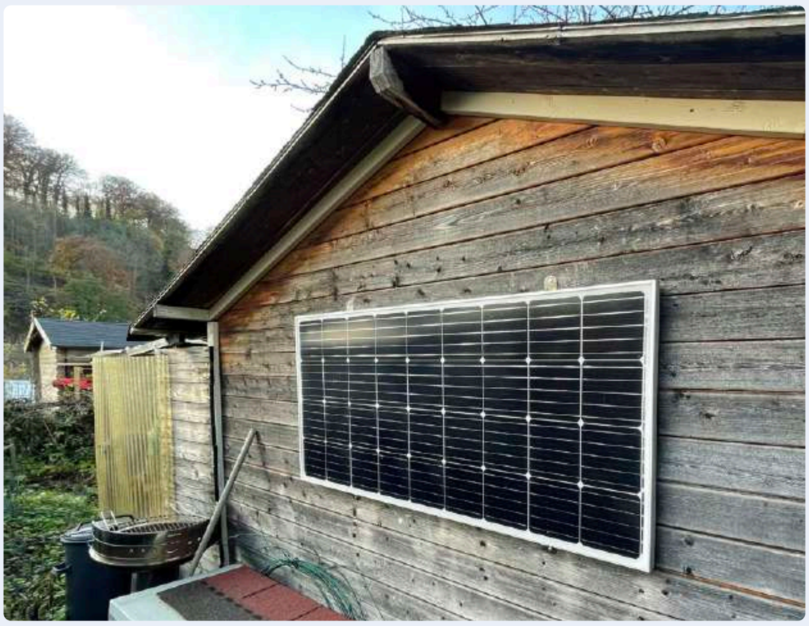


Figure 2: The 12 V solar panel mounted vertically on the wall of Klaus' shed. It has an output rating of 180 Wp.

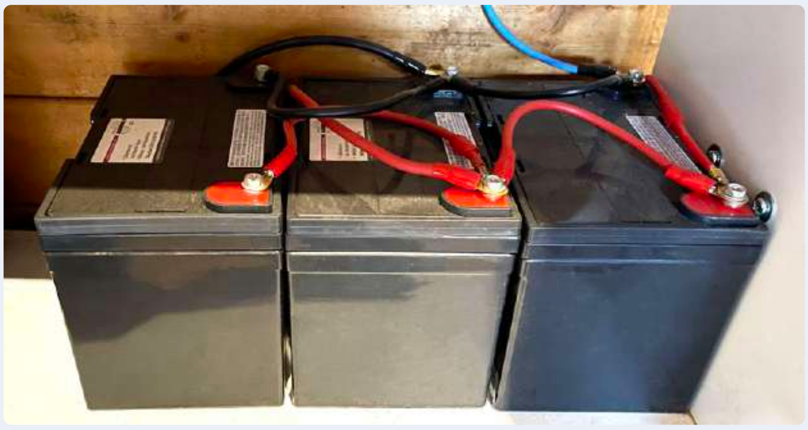


Figure 3: Three 12 V lead gel batteries, each rated at 36 Ah, are wired in parallel to act as energy storage in Klaus' shed.

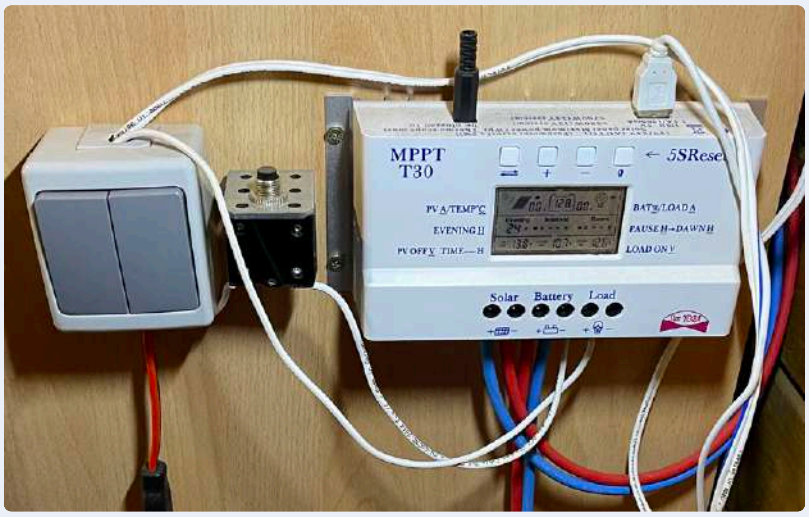


Figure 4: From left to right: light switch, 30 A electromagnetic circuit breaker, and MPPT charge controller.

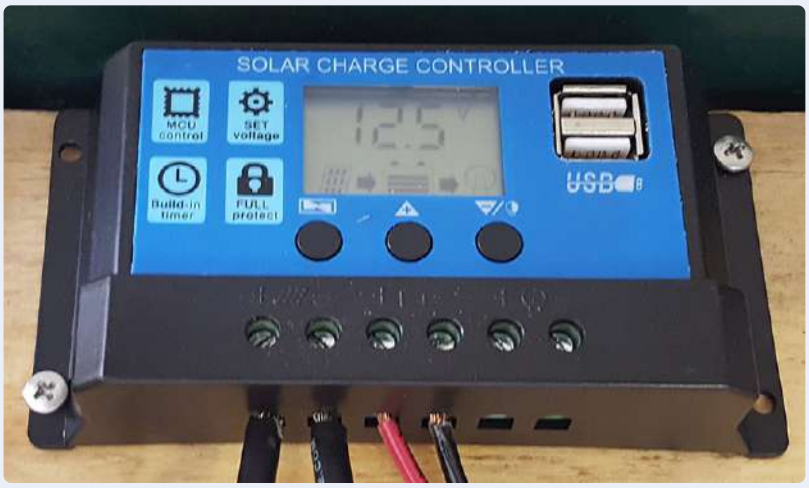


Figure 5: Charge controllers that look like this one certainly do not have MPPT functionality (even if they carry a sticker bearing those letters!). (Source: United States Department of Energy)

and requires careful management. Instead, Klaus plumped for a simple lead-based battery costing a fraction of that price. An obvious choice was a car battery, since these are designed for high peak currents. However, they have disadvantages: low efficiency, low cycle life and high self-discharge rate. For that reason he compromised on a gel battery: this type does not like high discharge currents, and so two batteries, each rated at 36 Ah, were connected in parallel. This combination offers a nominal 864 Wh of stored energy and cost a little under €150/US$170/£130.

In direct sunshine the selected solar panel delivers so much power that it can easily fully charge the batteries in a single day, and their capacity is enough to cover 1.5 days without sunshine. I had some misgivings about the high current draw at the input of the inverter, but Klaus decided to give it a shot and, if two batteries proved not up to the job, to buy another and add it in parallel. With the system installed and the batteries fully charged, a first test was fun using a 1 kW water heater. While running the voltage at the battery terminals fell to 11.7 V, but nevertheless it was easy enough to bring half a litre of water to the boil. At a rough estimate the efficiency of the battery (output energy divided by input energy) at these high currents is at most 50 % and the current does not do the battery's health any favours. A further 36 Ah battery was therefore ordered and wired in: see **Figure 3**. Now the initial terminal voltage when discharging at 85 A is a more acceptable 12.6 V, and the total capacity has been increased to nearly 1.3 kWh, guaranteeing over two days of reserve.

The Charge Controller

A search of eBay or of specialist distributors will turn up a wide range of charge controllers. Controllers rated at 10 A go for as little as €15/US$17/£13. However, a 180 Wp panel will deliver up to 15 A at 12 V, and so we need a controller rated at at least 20 A: these cost around €20/US$23/£17. If the charge controller is to be connected as shown in Figure 1, then it is better to choose a 100 A version, costing perhaps a little under €50/US$55/£45. Now let's go into the details.

The charge controller has the job of charging the battery using the power delivered from the panel, and terminating the charging process when a threshold voltage is reached. This ensures that the connected battery is not overcharged and hence damaged. Almost all charge controllers also control the load connection and ensure that the load is disconnected when a lower voltage threshold is reached, this time protecting the battery from deep discharge. They invariably employ a microcontroller and so most can be configured to work with different battery types, including lead-acid, lead gel and lithium chemistries. They also automatically adapt to a nominal terminal voltage of either 12 V or 24 V. Often it is also possible to configure the undervoltage and overvoltage thresholds manually.

The next consideration is the charger topology. All low-cost examples use PWM control, even if it says ‘MPPT’ on the device: labels are cheap, but a ‘real’ MPPT charger is better, more complicated and therefore rather more expensive.

In a PWM controller the charge current is adjusted so that the output voltage of the panel is just above the current battery terminal voltage. The battery is therefore charged at the maximum possible current given the amount of incident light on the panel and its size, and the state of charge of the battery over a wide range of conditions. The circuit to do this requires just a simple microcontroller and a power MOSFET: a low-cost solution, but not optimal.

Now, the output power of a panel is given by the product of its output voltage and output current. For any given panel and level of illumination there is a point where this product is maximized; almost invariably at this maximum power point the output voltage of the panel is above the battery voltage. An MPPT (maximum power point tracking) controller continuously determines where this ‘sweet spot’ is and drives a step-down voltage regulator circuit such that it draws the optimal current and delivers the maximum possible output power. In the best case the output power from an MPPT controller can be 30% higher than that from a PWM controller. However, this comes at a cost : even a low-cost MPPT controller will set you back over €50/US$55/£45, and a name-brand unit will be at least €100/US$110/£85. The 30 A charge controller shown in **Figure 4** is a low-cost MPPT type, and costs about €60/US$70/£50, but Klaus felt that the extra output power was worth the money. If you are looking for an MPPT controller, avoid ones like that shown in **Figure 5**: this type is available in a range of colours and with different markings.

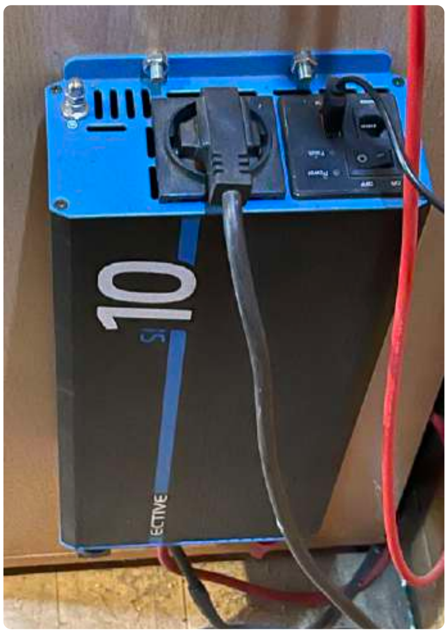


Figure 6: This 1 kW inverter made by Ective has proved very stable and reliable over time.

The Inverter

If it is a requirement to generate a 230 V AC output, then an inverter is essential. Low-cost examples with implausible power specifications and output voltage waveforms at best distantly related to a sinusoid are best given a wide berth. An important point to note is that the specification for maximum continuous

Advertisement

Bring your **creativity** and win an **Apple iPad Pro**

WizFi360 Design Contest

at maker.wiznet.io

- Official Wi-Fi Shield on ARM Open-CMSIS-Pack and Keil Studio Cloud
- Easy-to-connect Wi-Fi to Pico RP2040
- Azure Certified / Supports AWS SDK Examples

WizFi360 with

ARMKEIL

Raspberry Pi RP2040

ARDUINO

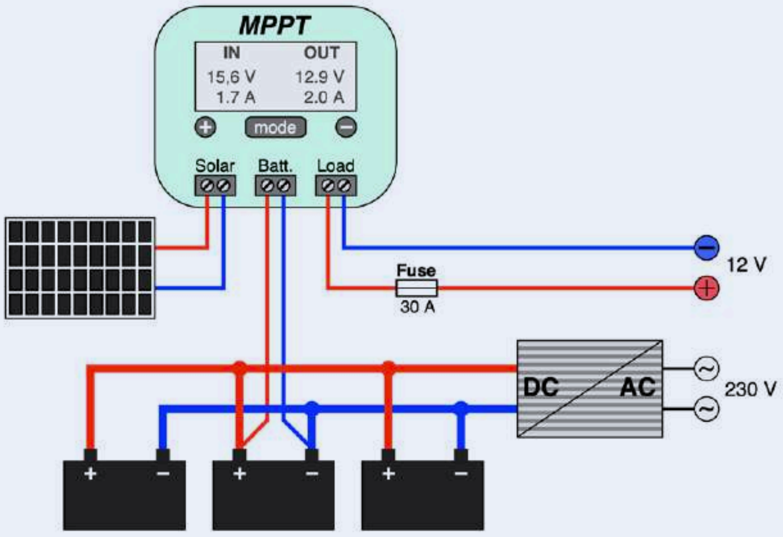


Figure 7: In Klaus' installation the inverter is connected directly to the rechargeable battery, and the 12 V output of the charge controller is provided with extra protection.

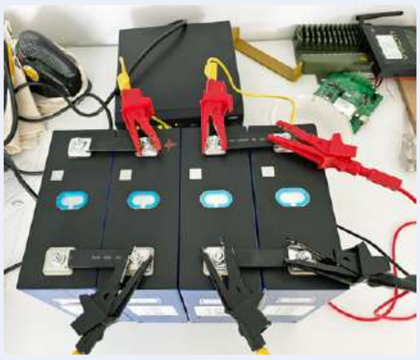


Figure 8: $LiFePO_4$ batteries under test before installation in Martin's boat. (Source: Martin Jepkens)

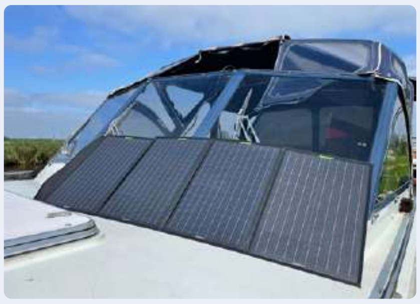


Figure 9: The foldable solar panel chosen by both Martin and Detlev can be stored below deck when underway. It is rated at 120 Wp. (Source: Martin Jepkens)

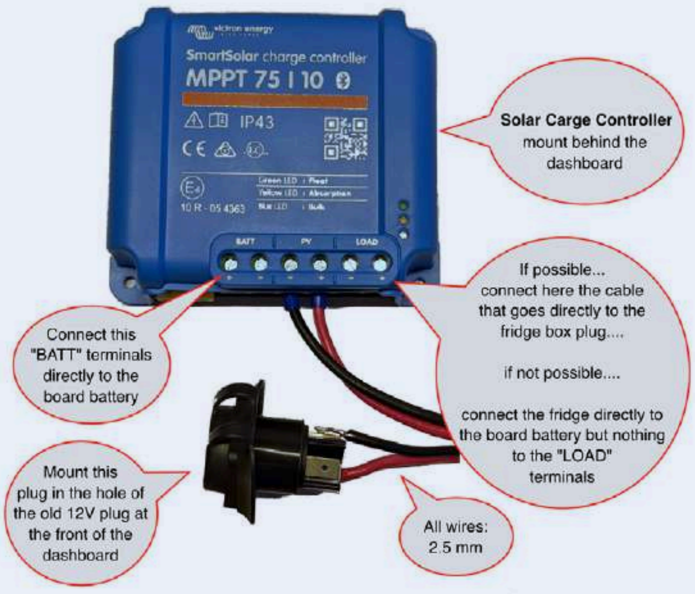

SmartSolar charge controller
MPPT 75 | 10
Solar Carge Controller mount behind the dashboard
If possible... connect here the cable that goes directly to the fridge box plug....
if not possible....
connect the fridge directly to the board battery but nothing to the "LOAD" terminals
Connect this "BATT" terminals directly to the board battery
Mount this plug in the hole of the old 12V plug at the front of the dashboard
All wires: 2.5 mm

Figure 10: Guide to installing the charge controller in Detlev's boat.

power is given for an ohmic load. Klaus' 1 kW inverter is perfectly suitable for driving a 1 kW 500 ml water heater; but the situation with an inductive or, less commonly, capacitive load is completely different. In this case we must also check the reactive power: note that the apparent power is always at least as great as the true power. Among the most problematic devices are electric motors, which can exhibit high inrush currents that will trigger the built-in overload protection circuitry on an inadequately-rated inverter. An allowance of 100 % when running motors is not excessive even for a high-quality inverter. Klaus' 1 kW inverter, shown in **Figure 6**, can comfortably handle an electric drill and a water pump rated at 450 W. It cost over €200/US$225/£170.

Wiring

As you may have surmised from the images so far, the wiring between the solar panel, charge controller, and battery is done using multi-stranded wire with a 6 mm^2 cross-section; the connections to the 12 V cigarette lighter sockets are not shown. The parallel connections between the batteries themselves are made using 16 mm^2 wire. The thicknesses of wire used need to be appropriate for the currents to be carried: this is not a good place to try to save money.

The inverter is connected directly to the battery using 16 mm^2 wire to minimize losses. A direct connection is only possible when the inverter (like the one here) offers undervoltage protection, switching off to protect the battery from deep discharge. The inverter is only enabled when the 230 V output is actually required: its quiescent current consumption, in the tens of milliamps, would otherwise be an unnecessary waste of energy. The final set-up is thus as shown in **Figure 7**.

Other Off-Grid Systems

The electricity supply in Klaus' shed is a typical example of an off-grid solar installation. Various suppliers offer ready-to-go packages comprising a solar panel, charge controller and inverter, with various nominal power levels. If you decide to opt for wind energy rather than solar, then again suitable generators and charge controllers are available based on broadly the same principles. For my part, last year I modified my robot lawn mower for autonomous power [2]. This needed just a 50 W panel and a simple PWM charge controller; no inverter was necessary. Since then I upped the battery capacity from 12 Ah to 30 Ah to help cover the periods of rainy weather we have had. I have also recently replaced the PWM charge controller with a better (and more expensive) MPPT controller, and the system can now generate enough electricity to mow the lawn even late into autumn.

There are of course many other applications for off-grid power. Two of my other friends have boats: Martin navigates his large steel-hulled boat through the riverscapes of the Netherlands, while Detlev makes mischief in the Med in his sportsboat with a planing hull. Both often spend days away from a harbour or other mooring where electricity is available and therefore would like to have more independence, especially as far as refrigeration is concerned: not in this instance just for beer, but for other sustenance as well. It would be ecologically unfriendly, not to mention inefficient, to run the engine frequently in order to charge the on-board battery, and so both have installed solar systems.

Now Martin is a smart engineer and doesn't need to rely on any advice from me. Nevertheless he discusses his ideas with me from time to time. He was wondering whether the generator on his boat could be overloaded if he happened to connect a huge 200 Ah $LiFePO_4$ battery across it. The dangers of such a course of action are explained in a YouTube video [3]. **Figure 8** shows how he set up his batteries for capacity testing: he decided on a battery with a $LiFePO_4$ chemistry mainly because of their long cycle life, but also because of their compactness compared to lead-based batteries. In Martin's boat the on-board battery is separate from the starter battery. In order to ease the burden on the alternator, the batteries are each charged via their own charge controller when the engine is running. A 120 Wp foldable solar panel and charge controller are also fitted for charging when underway (see **Figure 9**).

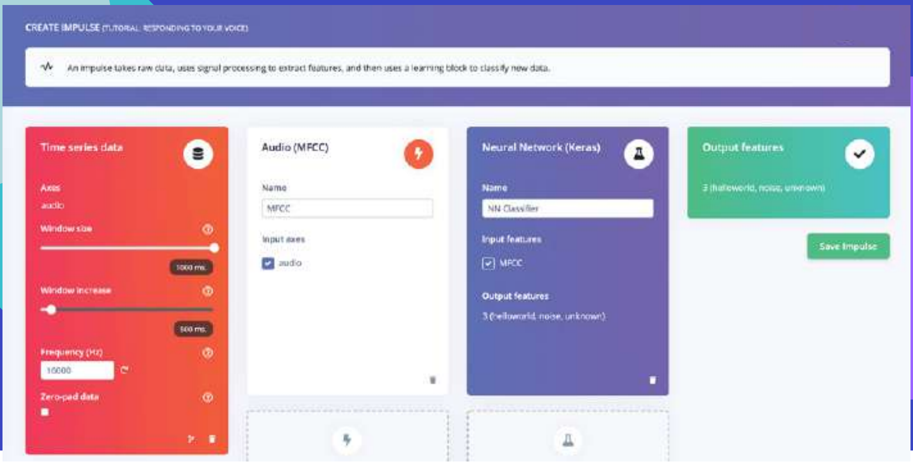
Because of the lack of available space, a fixed solar panel installation is not a practical proposition on Detlev's sportsboat. He therefore decided to use the same type of solar panel as Martin, although neither knew what the other had chosen! Detlev is not an electronics specialist, and at first he wanted to use his extra 120 A on-board battery because it was still rather new. I did some calculations for him and advised him that using the cigarette lighter socket on the 'bridge' of his vessel to connect the solar panel was not a great idea from a reliability point of view: I suggested the use a a waterproof Neutrik connector instead. I pre-wired the connector and drew up an installation guide (**Figure 10**) for his boatbuilder, so that the whole system could be set up in harbour in Istria. The combination of

Advertisement

EDGE IMPULSE

Advanced ML for every solution

Try free at edgeimpulse.com


CREATE IMPULSE (TUTORIAL: RESPONDING TO YOUR VOICE)

An impulse takes raw data, uses signal processing to extract features, and then uses a learning block to classify new data.

Time series data
Axes: audio
Window size: 1000 ms.
Window increase: 500 ms.
Frequency (Hz): 16000
Zero-pad data

Audio (MFCC)
Name: MFCC
Input axes: audio

Neural Network (Keras)
Name: NN Classifier
Input features: MFCC
Output features: 3 (helloworld, noise, unknown)

Output features
3 (helloworld, noise, unknown)

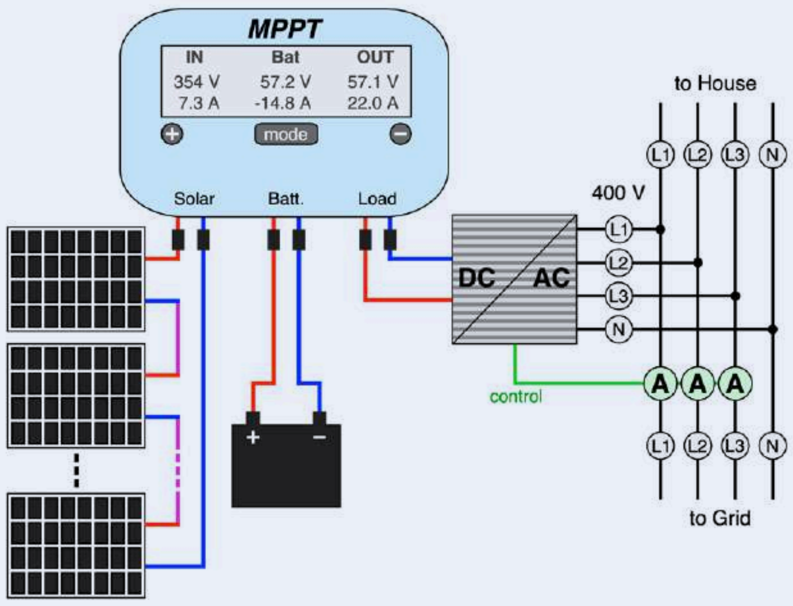


Figure 11: Semi-off-grid solar installation for a house. The three-phase current measurement controls the inverter in such a way that no electrical energy is driven into the grid.

foldable 120 Wp panel plus a Victron Energy MPPT charge controller came to a total of just under €500/US$550/£450. The charge controller has a Bluetooth connection and all parameters and graphs can be monitored using a smartphone app.

The Semi-Off-Grid House

In these times of dwindling feed-in tariffs interest has grown in a version of the fixed solar installation that as far as possible dedicates all of the generated energy to the demand of the house. An array of, for example, 10 modern solar panels will generate some 3.75 kWp; a suitable MPPT solar charge controller could then charge a $LiFePO_4$ battery with a capacity of say 6.5 kWh; and then a three-phase inverter could be controlled using a current monitoring circuit (the three ammeters towards the bottom right of **Figure 11**) to ensure that under no circumstances is any electrical energy driven into the grid. All the 'current' is therefore locally used. With electricity in Europe in 2022 costing around €0.35/US$0.40/£0.30 per kWh this is a very attractive option: not only does it avoid having a complex and also expensive grid-tied inverter with integrated charging electronics for the battery, it also avoids a large amount of bureaucracy: not a negligible consideration! (Note that such an arrangement may not be legal in all countries.)

In the arrangement in Figure 11 the savings from using the simplified design add up to €1000/US$1150/£850 to €2000/US$2300/£1700. It would take a few years to make up for that using a feed-in tariff. The most costly part of the system is the battery: a 6.5 kWh $LiFePO_4$ battery costs over €3000/US$3400/£2600. With a guaranteed 6000 charging cycles at 90 % discharge depth that means that around 36 MWh of energy will have flowed via the battery, making the battery cost around €0.09/US$0.10/£0.08 per kWh. Furthermore, at that point the battery is still not completely dead and so the effective cost per kWh will be even lower. Over the life of the battery a solution like this can save costs of very roughly €13000/US$15000/£11000 using energy one has generated locally. If an electric vehicle is also charged (at a low charge rate) then an installation of this kind can pay for itself within a few years.

210644-01

Contributors

Text and figures (unless otherwise stated): **Dr. Thomas Scherer**
Editor: **Jens Nickel**
Translation: **Mark Owen**
Layout: **Harmen Heida**

Questions or Comments?

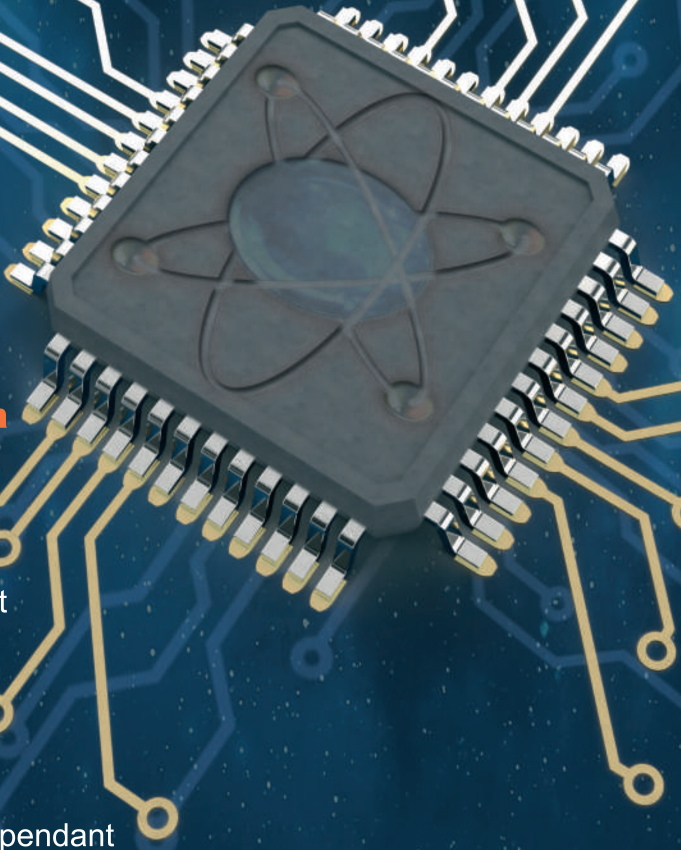
Feel free to send technical questions to the Elektor editorial team by e-mail to editor@elektor.com.

WEB LINKS

[1] T. Scherer, "Balcony Power Plant," Elektor Magazine September/October 2021: www.elektormagazine.com/210326-01
[2] T. Scherer, "Solar Power for Mowing Robots," Elektor Magazine July/August 2021: www.elektormagazine.com/200553-01
[3] Victron Energy, "How to not blow up your alternator when charging lithium," YouTube: www.youtube.com/watch?v=jgolocPgOug

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

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

Pre-Production Checklist

Set of board tests before Gerber Output

Includes placement, connectivity and clearance testing

Completely independant code for clearance checks

labcenter Electronics

www.labcenter.com

info@labcenter.com

+44 (0)1756 753440