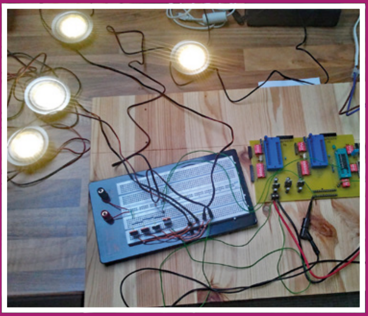
Edition 3/2017 | May & June 2017
www.elektormagazine.com

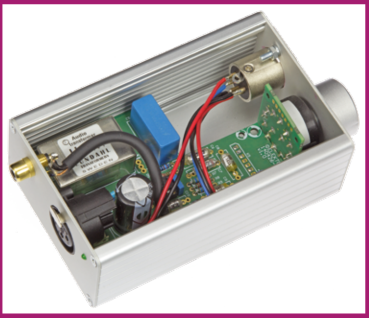
elektor

LEARN DESIGN SHARE



BBC micro:bit for Electronicists
In bed with mbed

Get on Board with IoT
Boards and kits for Internet of Things development

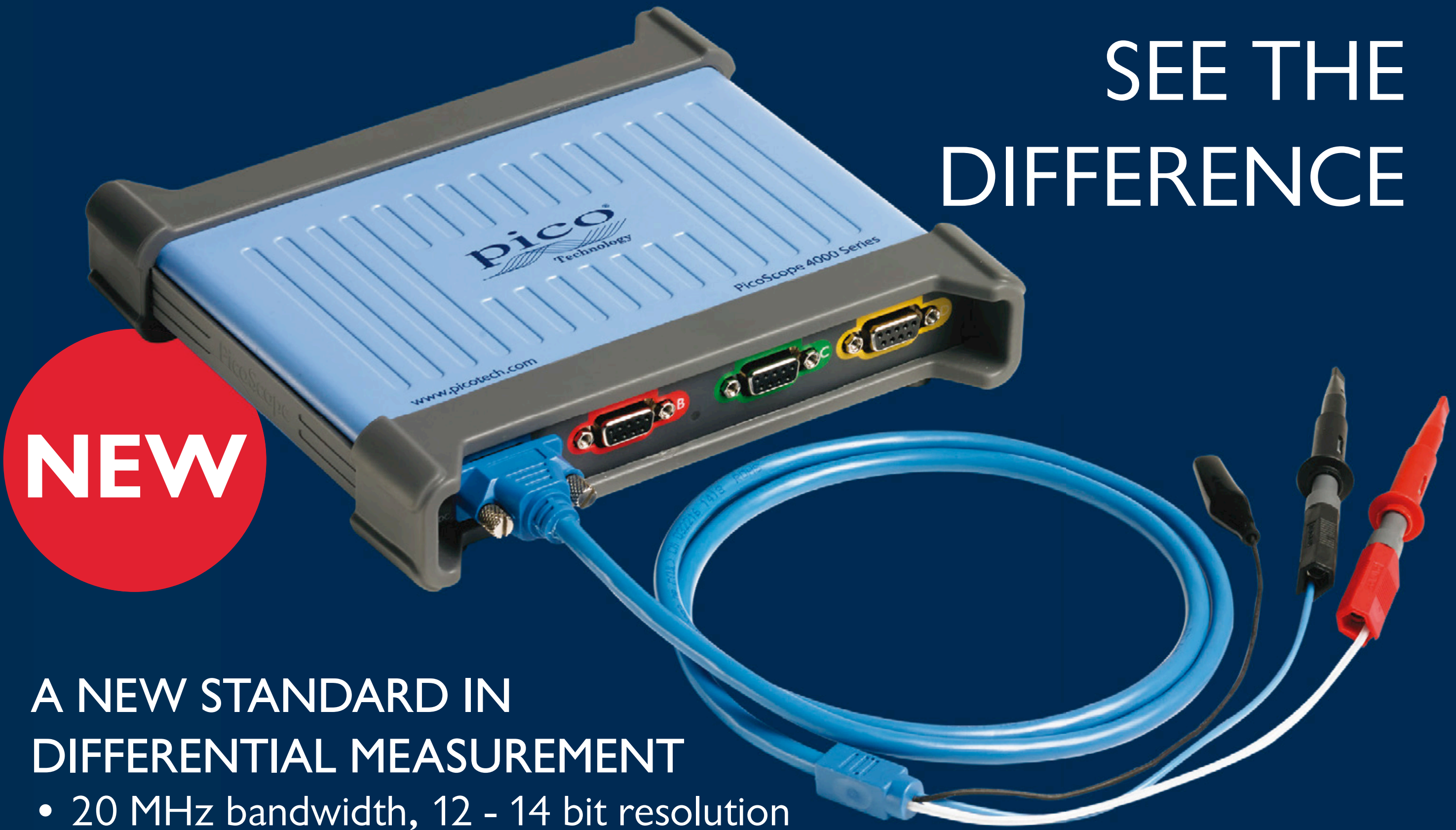


Microphone Preamp
with professional specs

4-Channel, 2-Flavor Remote Control

with XBee or nRF24 radio modules

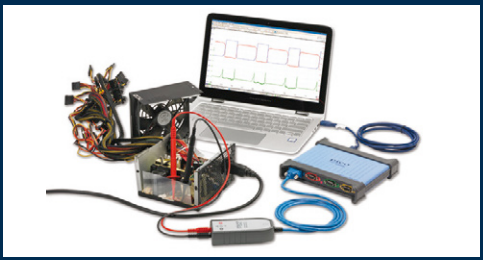
THE PICOSCOPE 4444



A NEW STANDARD IN DIFFERENTIAL MEASUREMENT

- 20 MHz bandwidth, 12 - 14 bit resolution
- 4 fully differential inputs
- 1000 V CAT III probes
- Low voltage probes and current clamps

With four true differential inputs, 12- to 14-bit resolution and wide differential and common-mode voltage ranges, the PicoScope 4444 and its accessories offer accurate and detailed measurement for a multitude of applications, from low-amplitude biomedical and electronic uses to 1000 V CAT III design and test.



For more information please visit www.picotech.com/PS502

Elektor Magazine

Edition 3/2017
Volume 43, No. 483
May & June 2017

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

www.elektor.com
www.elektormagazine.com

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

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

Head Office:
Elektor International Media b.v.
PO Box 11
NL-6114-ZG Susteren
The Netherlands
Phone: (+31) 46 4389444
Fax: (+31) 46 4370161

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

Advertising & Sponsoring:
Johan Dijk
Phone: +31 6 15894245
E-mail: johan.dijk@eimworld.com

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

Surfin’ the wave

Some say it’s not electronics at all and call it *disposable intelligence on a printed circuit board*: I mean the computerettes called Raspberry Pi, Arduino, micro:bit, BeagleBone, as well as a long list of elects, imitators and usurpers with slightly adapted names. All are different sea waves on which the Platform Generation is surfing to shore, some cautiously on low waves and struggling to stay upright on 8-bit boardlets, others showing off racing on waxed 64-bit boards, one ARM up and constantly defying Mr Gravity. All out there up on the waves are constantly helped by a favorable wind called the Internet with occasional gusts from the IoT. They appear not in the least bothered by safety concerns over the stability of their boards, as much of that is just a swirl under water. Speed and smooth rides is what they want.

A great spectacle it may be to observe this wave riding from the relative safety of the shoreline, one cannot help but wonder if the surfers did the tweaking, polishing and carrying of their boards themselves, or if they are able to exploit them to their full potential. As polished and fast as Surfboard-64 may seem, to an electronics engineer the fun is also in understanding its secret, and finding ways to customize it — to beat others.

Too many users of the Great Boards above never progress beyond running out of the box applications developed by others. They are comfortable with everything programming, but the physical edge of the board, where the connectors are, induces fear of dark things like sensors and solder tin, or inductors!

I call on you, Elektor readers, the Enlightened Ones with sacred knowledge of hardware & sensors down to IC pin levels and elusive microvolts, to assist the Platform Generation in overcoming their false fears of real-world electronics. I have the curriculum ready for you. You are holding it right now, it is 132 pages of paper (or a 1.5 Mbyte pdf download) packed with leads and trigger signals. If your student(s) baulk and are off to the beach with a board or two, feel free to enjoy the articles yourself.

Jan Buiting, Editor-in-Chief

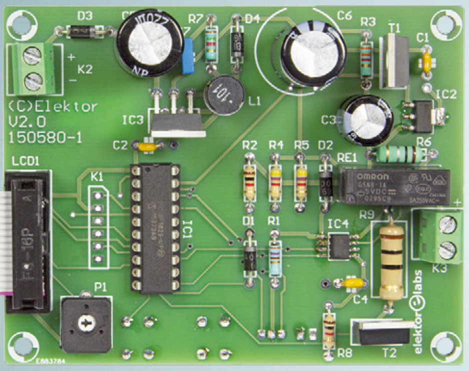
The Circuit

Editor-in-Chief:	Jan Buiting
Deputy Editors:	Thijs Beckers, Clemens Valens
Translators:	David Ashton, Jan Buiting, Martin Cooke, Ken Cox, Arthur deBeun, Andrew Emmerson, Tony Marsden, Mark Owen, Julian Rivers
Membership Manager:	Raoul Morreau
International Editorial Staff:	Thijs Beckers, Mariline Thiebaut-Brodier Denis Meyer, Jens Nickel
Laboratory Staff:	Ton Giesberts, Luc Lemmens, Clemens Valens, Jan Visser
Graphic Design & Prepress:	Giel Dols
Publisher:	Don Akkermans

This Edition

Volume 43 – Edition 3/2017
No. 483 May & June 2017

118

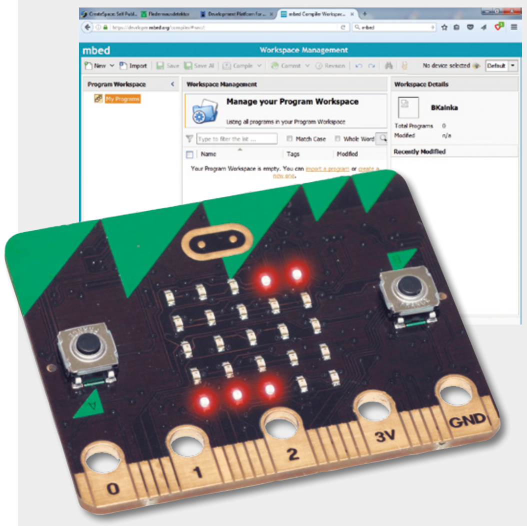


Regulars

BBC micro:bit for Electronicists

In bed with mbed

20



The BBC micro:bit 'computerette' is not only for school-age users — it's also a great controller board for grown up electronicists. Crammed into its tiny footprint is almost everything that you need: digital inputs/outputs with PWM and A-D converters along with a USB interface for power supply, programming and transferring data. A variety of sensors and Bluetooth round off the configuration.

Features

Projects

6

Boards and kits for Internet of Things development

Li-Ion charger

to charge the cell correctly

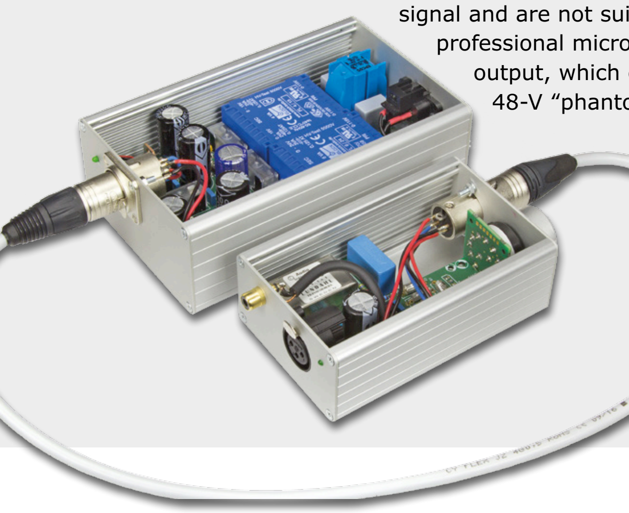
- Charges any Li-Ion battery with a voltage rating of 3.6 V.
- Formatting mode to resuscitate cells in deep-discharge.
- Discharge mode displays the true value of battery capacity.
- All the important information is shown on an LC display.

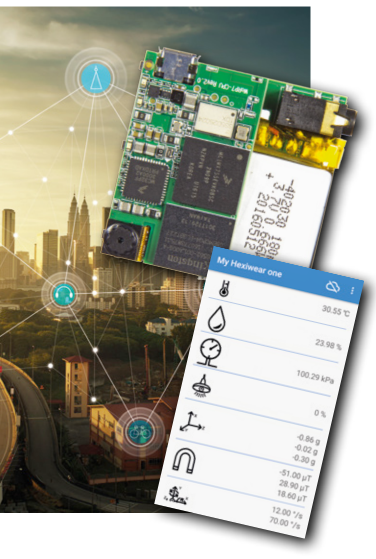
Microphone Preamp

28

with professional specs

Tape recorders have been obsolete for ages now, replaced with laptops fitted with a good-quality 'USB sound card'. Now while a PC's 'line' input generally offers acceptable characteristics, 'mic' inputs typically have only mediocre noise and distortion levels. Above all, they offer only a mono signal and are not suitable for connecting professional microphones with a balanced output, which often also require 48-V "phantom" powering. Here I'm suggesting a preamplifier that gets around these two shortcomings and is bound to please those who own 'top range' sound cards.





Next Editions

Elektor Magazine 4/2017

Sniffer for RFM12 • 1-2-4-8- Attenuator • Voltera PCB Printer • PCB Manufacturing • Vacuum Tube Curve Tracer • RPi High-End DAC • 7-Segment Scoreboard • nWatch • LoRa with ST Boards • MicroPython Project • Digital AF Experiments • techBASIC for iPad/iPhone • 2-Wire LED & Lamp Dimmer • and much more.

Elektor Magazine edition 4 / 2017 covering July & August is published on 21 June 2017. Delivery of printed copies to Elektor Gold Members is subject to transport.

Elektor Business Magazine 3/2017

Sensors & Measurement Technology

Functional Safety & Cyber Security (TüV Rheinland) • Electromotor Maintenance (Rheintacho) • IoT Gateway & Sensor Sticks (Binder) • Active-Sensor Powering (Linear Technology) • Measurement Uncertainty (Mettler Toledo) • Remote I/O Configurations for Monitoring Systems (AMC Systems) • Ultralow Power Accelerometer (Analog Devices) • 500 kHz / 1 MHz Bench LCR Meter (B & K Precision) • LabVIEW for Li-Ion Battery Electrolyte Filling & Vacuum Sealing (National Instruments) • and more.

Elektor Business Magazine edition 3/2017 is published on 17 May 2017. Contents and article titles subject to change.

Get on Board with IoT

Boards and kits for Internet of Things development

By **Viacheslav Gromov** (Germany), readers@gromov.de

The Internet of Things (IoT) is one of the most popular topics among our readers, especially when it comes to intelligent home automation. The first step in developing your own IoT system is to set up a rapid prototyping environment, and to this end embedded systems providers offer a wide range of astonishingly low-cost development kits, containing everything you could possibly wish for to start you on your way. Even a beginner can get a first prototype up and running within minutes.

Modern IoT development kits contain pretty much everything you need to get started, from the USB cable for connecting up the board and a selection of sensors right the way through to a battery and an enclosure. But despite the richness of these offerings it is very easy to get lost in the maze of possibilities offered by the Internet of Things: Which processor should I use? Which sensors? Which radio technology? What sort of power supply? And all that is before we come to consider essential aspects such as optimizing for cost, development time, power consumption and anything else you can think of. The burgeoning (although still not saturated) market is chock-a-block with platforms representing every technology and application area.
In this necessarily incomplete overview we will not consider familiar IoT platforms such as the Micro:Bit, Arduino and its cousins, the vast array of ESP8266-based boards and other more or less professional products. Instead we will concentrate our attention on solutions offered by the large semiconductor manufacturers, as it is surely they that are responsible for the rapid proliferation of IoT technologies. The survey on the following pages will range from processors to sensors and wireless devices, whether brand-new or established, familiar or relatively unknown, low-cost or expensive, general-purpose or application-specific. We will look at the particular strengths of each of the products we have selected relevant to IoT applications, and examine the characteristics of the hardware and of the software. We hope that there will be something for everyone!

Hexiwear by MikroElektronika and NXP

NXP (formerly Freescale) has of course produced its own Internet of Things development boards, but more interesting perhaps are the alliances it has formed with other manufacturers and with distributors, and it is of such an alliance that Hexiwear (**Figure 1**) [1] was born. The platform is produced and marketed by MikroElektronika, while NXP is responsible for the microcontrollers and most of the sensors. Moreover, NXP provides considerable support for the platform, for example in making adaptations to Kinetis Design Studio. But what exactly is Hexiwear? Physically it is a 'wearable' in the form of a large wristwatch, available with a variety of enclosure types and bands. It can also be used as a stand-alone device with a wide range of expansion options. The Hexiwear board is very versatile, with options including BLE, USB, sensors (for heartbeat, light, temperature, pressure, humidity, linear and angular acceleration, and magnetic field), a 1.1 inch color OLED display and a 190 mAh rechargeable battery that in normal use will have a life of many hours. The device can be controlled using six touch elements on its front surface, and the built-in vibration motor can be used to provide haptic feedback. The front of the device also features a programmable RGB LED.

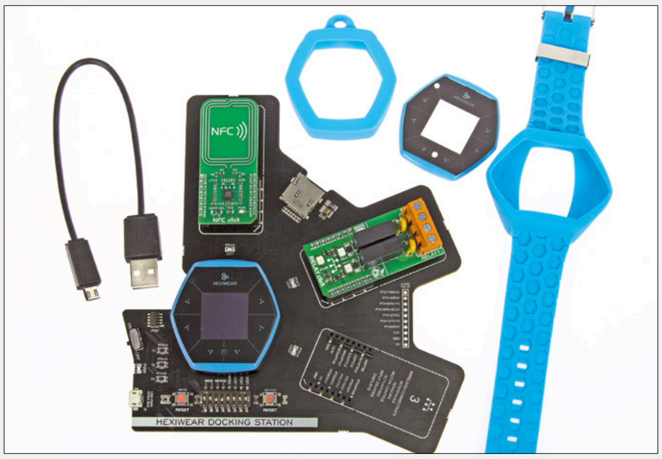
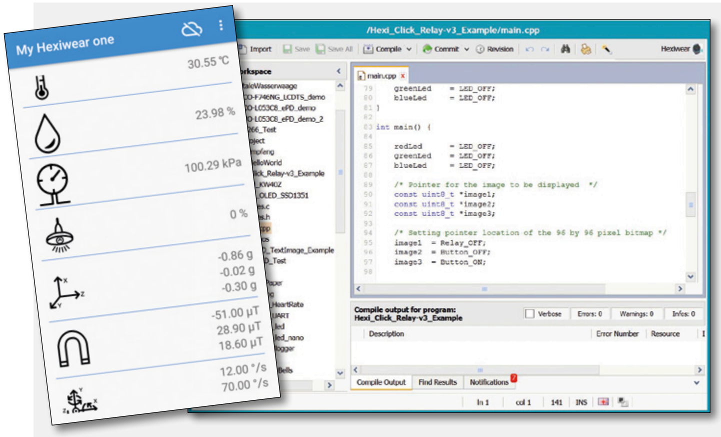


Figure 1. The Hexiwear on its docking station with two of the click boards available for expansion. Accessories include a USB cable and the add-on Color Pack.

Inside there are two microcontrollers beavering away: the familiar K64F (M4F, 120 MHz, 1 Mbyte flash), which is responsible for the main control functions and which looks after all the connected peripherals, and a KW40Z (Cortex-M0+, 48 MHz, 160 Kbyte flash) from the range of dual-mode (Bluetooth and IEEE 802.15.4) radio devices, which provides the Hexiwear with its BLE (Bluetooth Low Energy) connection. The board sports a 50-way connector on its underside which carries the most important interfaces, and which is designed to mate with the (separately supplied) docking station board. This latter board provides three user buttons and LEDs, reset buttons for the two microcontrollers, a microSD card slot, and I^{2}S, micro-USB and SWD interfaces. In the middle of the docking station board there are three slots which are compatible with the popular MikroElektronika click boards. Hundreds of different click boards are available with various sensors, actuators and interfaces. Using these click boards it is therefore possible to build prototypes rapidly and economically.
As supplied Hexiwear is programmed with firmware that demonstrates all its sensor functions. It also allows the device to be connected to a smartphone over Bluetooth using the WolkAbout Hexiwear app: this app allows you to monitor data in real time, upload them to the WolkAbout cloud, and visualize them in various ways. Firmware updates can be carried out over the wireless interface ('OTAP', or over-the-air programming), but it is also possible to program the Hexiwear over its micro-USB interface, using an OpenSDA debugger as a bridge to the microcontrollers. There are three options [2] for writing your own software to run on the Hexiwear: for beginners the simplest approach is to use the mbed online environment (**Figure 2**); Linux aficionados might prefer to use the Zephyr distribution; and those



```
    greenLed    = LED_OFF;
    blueLed     = LED_OFF;
}

int main() {

    redLed      = LED_OFF;
    greenLed    = LED_OFF;
    blueLed     = LED_OFF;

    /* Pointer for the image to be displayed  */
    const uint8_t *image1;
    const uint8_t *image2;
    const uint8_t *image3;

    /* Setting pointer location of the 96 by 96 pixel bitmap */
    image1  = Relay_OFF;
    image2  = Button_OFF;
    image3  = Button_ON;
```

```
Compile output for program:
Hexi_Click_Relay-v3_Example
Verbose   Errors: 0   Warnings: 0   Infos: 0
Description                              Error Number   Resource
```

Figure 2. The screenshot on the left shows the Hexiwear app with real-time data. On the right is the mbed environment, illustrating the clear and simple structure of the program code.

who wish to make the most of the possibilities offered by the hardware can use Kinetis Design Studio (KDS). A wide range of Hexiwear IoT projects can be found at hackster.io to provide further inspiration. The documentation, all open-source, and software support are of a very high standard. The basic Hexiwear board comes in at about US$50, rising to about double that when you add on the optional docking station and enclosure.

While on the subject of NXP, we should mention the tiny WaRP7 board (**Figure 3**), produced in cooperation with Farnell element14 [3]. This board is supplied with a 180 mAh rechargeable battery and offers USB as well as NFC, BLE and WLAN wireless interfaces. There is also a camera and many other sensors on the board. The processor is an i.MX 7 Solo (Cortex-A7 plus Cortex-M4) augmented with external memory chips, and there is also a slot for a click board. A circular color touch display can be added. The platform is recommended for developers who are comfortable working with Linux distributions and with the command line and who want to understand the system in depth. Alternatively, the Android operating system can be used. The price is around US$90.

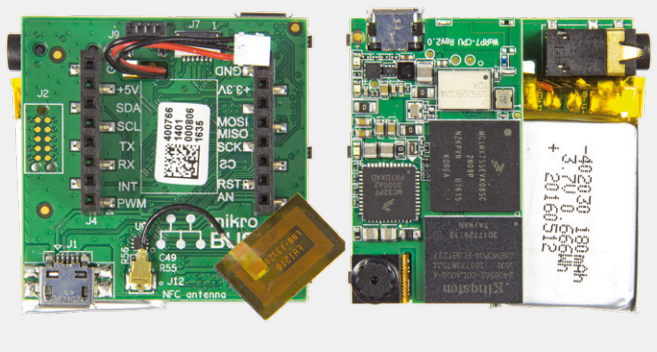


Figure 3. Front and back of the WaRP7 board.

STMicroelectronics SensorTile

STMicroelectronics (ST) has recently brought to market an very interesting kit called SensorTile [4], priced at around US$80 (**Figure 4**). The device is the size of a coin and is based on an STM32L476 microcontroller (Cortex-M4F, 80 MHz, 1 Mbyte flash). It includes a BlueNRG-MS device for BLE communications, a microphone, a barometer, an accelerometer, a gyroscope and a magnetic field sensor. The most important connections are brought out to solder pads on the edge of the board as well as to a small connector on the bottom side of the board.

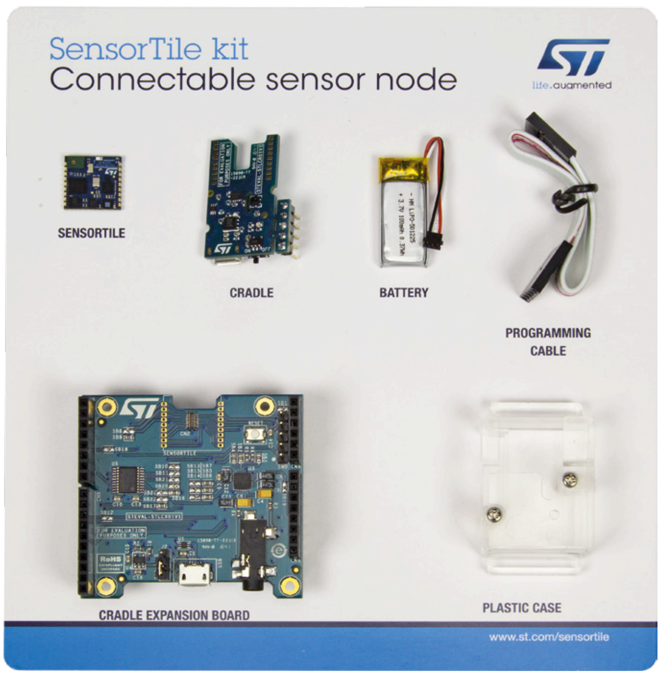
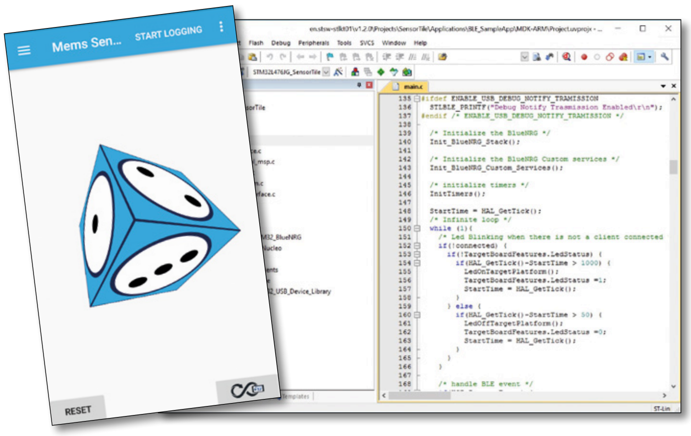


Figure 4. The SensorTile kit and its accessories.



```
#ifdef ENABLE_USB_DEBUG_NOTIFY_TRAMISSION
  STLBLE_PRINTF("Debug Notify Trammission Enabled\r\n");
#endif /* ENABLE_USB_DEBUG_NOTIFY_TRAMISSION */

  /* Initialize the BlueNRG */
  Init_BlueNRG_Stack();

  /* Initialize the BlueNRG Custom services */
  Init_BlueNRG_Custom_Services();

  /* initialize timers */
  InitTimers();

  StartTime = HAL_GetTick();
  /* Infinite loop */
  while (1){
    /* Led Blinking when there is not a client connected
    if(!connected) {
      if(!TargetBoardFeatures.LedStatus) {
        if(HAL_GetTick()-StartTime > 1000) {
          LedOnTargetPlatform();
          TargetBoardFeatures.LedStatus =1;
          StartTime = HAL_GetTick();
        }
      } else {
        if(HAL_GetTick()-StartTime > 50) {
          LedOffTargetPlatform();
          TargetBoardFeatures.LedStatus =0;
          StartTime = HAL_GetTick();
        }
      }
    }

    /* handle BLE event */
```

Figure 5. On the left, an example app demonstrating the accelerometer; on the right, an excerpt from the example code template in the Keil development environment.

The SensorTile cannot operate without a power supply, and so two adapter boards are included in the kit. The first adapter, called a 'cradle', must be soldered to the SensorTile board: it adds a humidity and temperature sensor, a microSD card slot, an SWD programming connection and a micro-USB charging connector. In the box there is also a 100 mAh rechargeable battery, which, when fitted to the cradle, will power the SensorTile. The whole unit can then be mounted in the transparent case also supplied with the kit.

The second adapter takes the form of an Arduino expansion shield that includes an audio DAC (with jack socket for output) as well as an SWD connection and a micro-USB connector. The shield can be used with the SensorTile in a stand-alone con-

figuration, or equally can be plugged into a Nucleo or other board that follows the Arduino layout. The latter option allows the SensorTile to be controlled from another microcontroller, or vice versa.
Both adapter boards feature an SWD connection which is used to program the SensorTile. This is most conveniently done using an ST-LINK programmer, which is present on all the 64-pin Nucleo development boards, and a suitable connector cable is included in the kit. ST has also produced an app called ST BlueMS, which demonstrates the capabilities of the SensorTile 'out of the box' (**Figure 5**). Many of the sensor examples require installation of a license before they can be used: the license is delivered automatically by e-mail, but the application process is relatively speedy and painless. The app also allows the upload of data to IBM's Watson IoT cloud.
Software can be built using any STM32-compatible compiler. You can use the (rather complex) BLUEMICROSYSTEM2 example as a guide when writing your own code: this is the firmware that is supplied ready-programmed into the SensorTile as supplied. Alternatively you can use the 'starter firmware' template which is trimmed down to just the basic functions, and which comes with three example programs to help you get started.

Cypress Semiconductor solar-powered IoT device kit

If this kit [7], which costs around US$50, contained just a Cypress BLE module, there would be little to make it stand out from the crowd of IoT devices. Admittedly the module is a good example of its kind, and the PSoC Creator development environment is highly suitable for this type of application, but the real showstopper is the solar cell and energy harvesting circuitry built into the board. Here we see a confluence of trends: IoT applications drive developments in power optimization, and these in turn prompt developments in energy harvesting. Energy harvesting is becoming an essential aspect of the Internet of Things.

The kit shows off the capabilities of the PRoC EZ-BLE module (CYBLE-022001-00, Cortex-M0, 48 MHz, 256 Kbyte flash). As

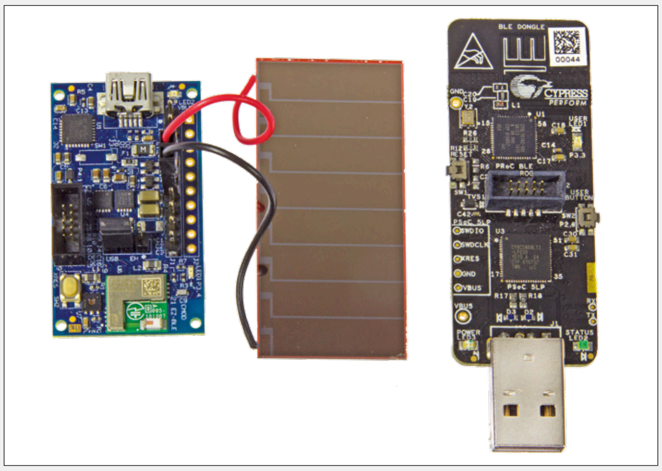
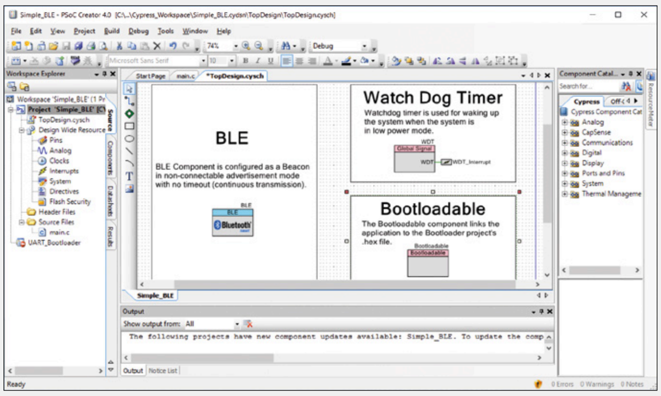


Figure 8. The two circuit boards and the solar cell featured in the IoT Device Kit.



Watchdog timer is used for waking up the system when the system is in low power mode.

BLE Component is configured as a Beacon in non-connectable advertisement mode with no timeout (continuous transmission).

The Bootloadable component links the application to the Bootloader project's .hex file.

```
The following projects have new component updates available: Simple_BLE. To update the comp
```

Figure 9. Thanks to the modular 'building block' approach even BLE programming is a simple task: everything is automatically pre-configured.

well as the module and the energy-harvesting circuit, the kit includes sensors for temperature, humidity and light on the board itself, and further sensors can be connected externally. The kit also includes a USB-to-BLE interface board which allows communication between a PC and the energy-harvesting board (see **Figure 8**). As well as data transfer, the board lets you monitor the signal strength so that you can assess the available range when using different energy sources (for example, a piezo generator rather than a solar cell). The PRoC on the energy-harvesting board can be programmed using an external debugger (such as the MiniProg3, not supplied) connected to its SWD port, and code can be written using the modular and straightforward PSoC Creator [8]: see **Figure 9**. The package also includes a couple of extra leaded components to support the example introductory projects.

Texas Instruments SimpleLink SensorTag

For around US$30 you can be the proud owner of a TI SensorTag (see **Figure 10**) [9]. On unpacking the kit, first impressions are a little underwhelming: a matchbox-sized dongle in a rubber enclosure. On second glance, however, it becomes clear what this little dongle has to offer: after installing TI's SensorTag app (see **Figure 11**) you can monitor all the tag's sensor data over BLE or WLAN (depending on the type of tag), upload them to IBM's Watson IoT cloud and process them.
One of the most recent tags in this series contains a CC1350 dual-mode microcontroller (Cortex-M3, 48 MHz, 128 Kbyte flash), which supports BLE as well as communications in the sub-1 GHz band over distances of up to 2 km. Externally all that is visible is a couple of buttons, but behind the narrow transparent opening in the inner plastic enclosure there is a wealth of sensors: a microphone, light, humidity, atmospheric pressure and magnetic field sensors, a standard temperature sensor and an infrared thermopile sensor, and a gyroscope and accelerometer. Plus, of course, a user LED. The low-power design allows the tiny button cell on the back of the circuit board to last for up to

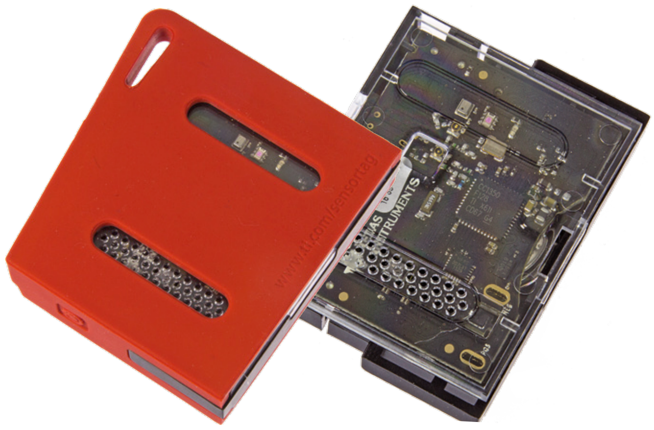


Figure 10. The CC1350 SensorTag lurks inside its enclosure.

a year, depending on the operational duty cycle.
That would all make for a useful device, even if the code programmed into the dongle at the factory could not be changed, but of course the possibilities do not stop there: the dongle can be programmed using the wireless-compatible version of TI's Code Composer Studio (CCS) [10] over the JTAG connector provided. Since the hardware is fully assembled, it is just a case of writing the software to make your first prototype. To help with this many libraries are available, and there is comprehensive documentation, including on how to program a smartphone app. There is also a range of additional software tools available, including Sensor Controller Studio, which can be used to configure sensor applications for the microcontroller. The SensorTags also support wireless programming (OTAP): for example, it is possible to update a tag's firmware over BLE from within an app. A very recent development is the ability to connect the tag to an expansion board called a 'DevPack' using its 20-way connector. DevPacks offer extra peripherals such as displays and RGB LEDs, and the range of functions available is growing all the time. There is little free space inside the rubber enclosure, and so to use a DevPack it is necessary to make a cut in the enclosure at the right point: not difficult, but a steady hand and a sharp knife are required.
Particularly recommended is the 'Debugger' DevPack, at around US$15. This allows a tag to be programmed conveniently from a USB port, obviating the need for larger-scale more expensive debuggers. Overall the price/performance ratio offered by this platform is excellent. A similar platform is also available using the CC2650 with BLE and ZigBee or 6LoWPan (supporting IEEE 802.15.4). And, according to the manufacturer, a WLAN tag is in the pipeline.

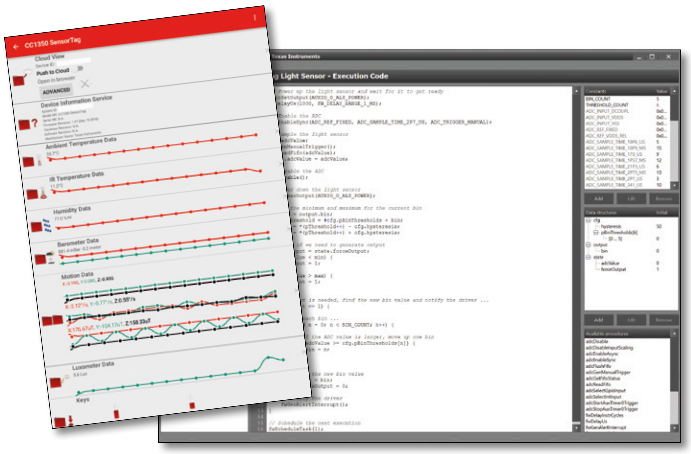


Figure 11. On the left, the main window of the SensorTag app; on the right, a screenshot from Sensor Controller Studio.

Bosch
Connected Devices and Solutions XDK

The Bosch XDK110 (**Figure 6**) has been on the market for longer, but nevertheless remains popular [5]. The high quality of the enclosure, accessories, circuit board and online infrastructure come at a price: around US$200. Bosch is recognized worldwide for its advanced sensors, and a good number of these are included in the this kit. The XDK itself comes in a plastic enclosure with four LEDs, two buttons and a transparent opening for the sensors (which cover magnetic field, angular and linear acceleration, light, temperature, humidity, atmospheric pressure and sound). Around the edge there is a slot for a microSD card, a programming connector and a 26-way header that interfaces to the 'T-board' using a cable (both supplied).
The T-board, or 'XDK gateway', can for example simply be plugged into a breadboard to make a first prototype, extending the XDK beyond its enclosure. You can then experiment with sophisticated control and monitoring functions: the ARM Cortex-M3 core with 1 Mbyte of flash memory provides plenty of processing power. A microSD card can be used if additional data storage capacity is needed, and the 560 mAh rechargeable battery allows for mobile applications with long battery life. The board offers BLE and WLAN interfaces, which can connect to other IoT devices or a smartphone: see the VirtualXDK example program and its app in **Figure 7**. It is also possible to set up a connection to the Internet. The unit can be programmed using the free XDK Workbench [6], which comes with many example programs and accompanying step-by-step instruction

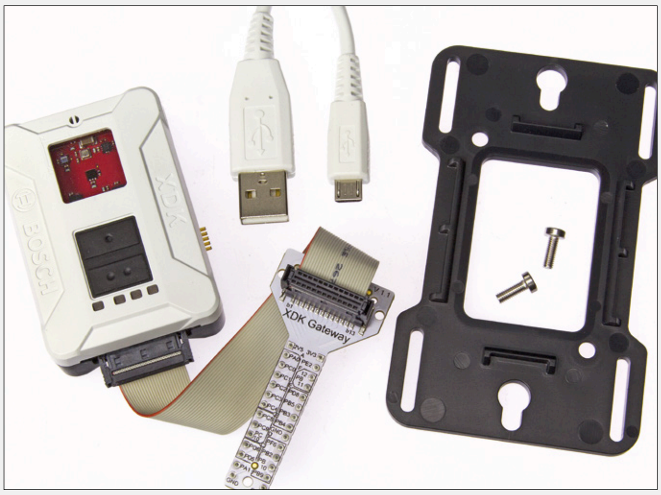
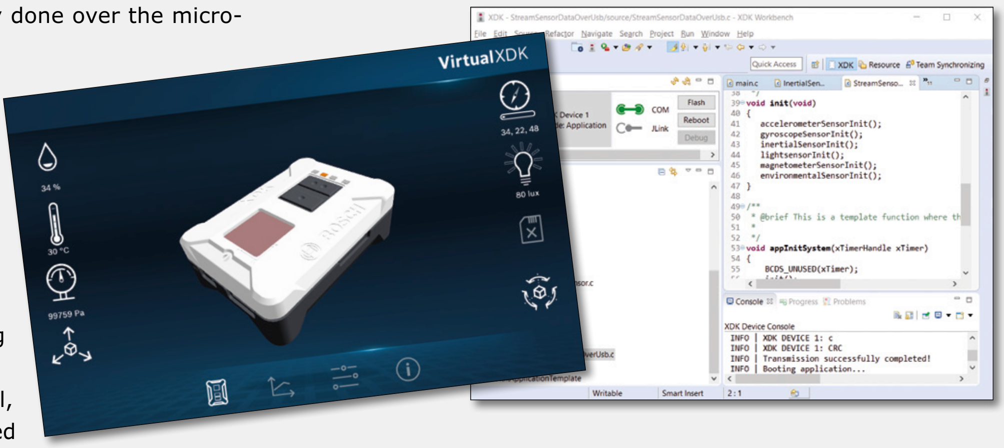


Figure 6. The main components of the XDK110 kit: even a wall fixing bracket is included. The photo shows how astonishingly small the device is.

guides. Programming is normally done over the micro-USB connector with the help of the bootloader that is supplied ready-programmed into the device; in some cases it will be necessary to resort to connecting an external debugger to the programming port provided. Other software suppliers, such as relayr, support the XDK through their own online platforms for cloud computing and data analysis.
In summary, this is a professional, well thought out and well supported platform aimed at industrial users, which despite its relatively large price tag offers good value for money. ⏮

(160343)



```
void init(void)
{
    accelerometerSensorInit();
    gyroscopeSensorInit();
    inertialSensorInit();
    lightsensorInit();
    magnetometerSensorInit();
    environmentalSensorInit();
}

/**
 * @brief This is a template function where th
 *
 */
void appInitSystem(xTimerHandle xTimer)
{
     BCDS_UNUSED(xTimer);
```

```
INFO | XDK DEVICE 1: c
INFO | XDK DEVICE 1: CRC
INFO | Transmission successfully completed!
INFO | Booting application...
```

Figure 7. On the left, the main view presented by the VirtualXDK app. On the right, the Eclipse-based XDK Workbench holds few surprises: only the programming procedure and use of the bootloader will be unfamiliar, but learning these will not present much of an obstacle to the experienced developer.

Some other options

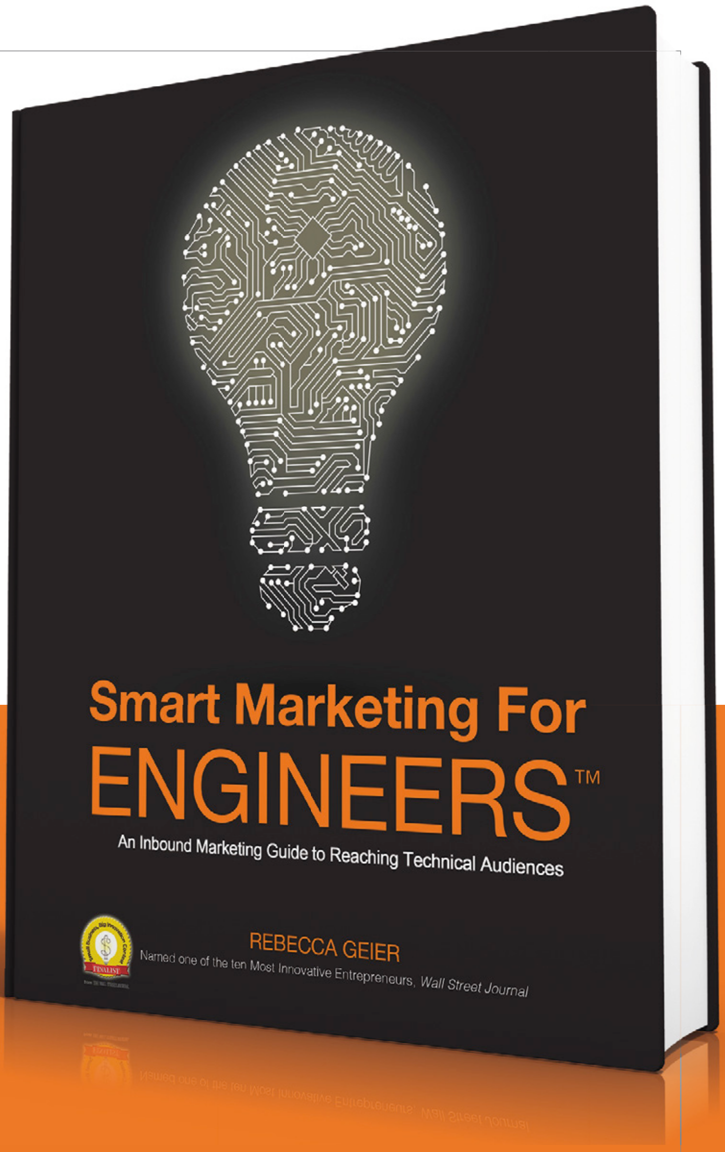
AT88CKECC-AWS-XSTK by Microchip, http://goo.gl/FPXVoi	This modular kit, coming in at around US$200, shows off Microchip's most recent IoT security technologies, working in conjunction with Amazon Web Services (AWS).
P-NUCLEO-LRWAN1 by ST, http://goo.gl/VLyLCT	This kit, at about US$40, comprises an L073RZ Nucleo board and a LoRa expansion shield, offering easy access to this trending IoT technology.
P-NUCLEO-USB001 by ST, http://goo.gl/wjDplk	USB-C is a common interface on modern IoT devices (even if only for charging). This F072RB Nucleo board with a power delivery expansion board, costing about US$50, demonstrates the possibilities.
C027 mbed-based IoT kit by u-blox, http://goo.gl/zZV21s	The focus of this kit is on GPS location and on UMTS and GSM communications, which are often requirements for smart devices. Depending on the version, the cost is around US$100.
Thunderboard React/Sense Kit by Silicon Labs, http://goo.gl/dfGhfy	These two tiny boards offer a wide range of sensors and other peripherals, and a choice of radio technologies depending on the model. At around US$30 they offer a good price/performance ratio.
NuMaker Uni by Nuvoton, http://goo.gl/Nm9XkG	A range of very low-cost IoT boards in the NuMaker series is available from this Chinese manufacturer. They include sensors, Bluetooth and WLAN, all for under US$30.
Synergy S3A7 IoT Fast Prototyping Kit by Renesas, http://goo.gl/ov4UsG	This modular IoT kit, costing some US$150, features a relatively large display. The 'Sandbox' technology allows the kit to be expanded using a range of different sensors and radio modules.
...and many more besides!	

Web Links

[1] www.hexiwear.com
[2] www.hexiwear.com/getting-started/
[3] www.element14.com/community/docs/DOC-79058/l/warp7-the-next-generation-iot-and-wearable-development-platform
[4] www.st.com/sensortile
[5] http://xdk.bosch-connectivity.com/
[6] http://xdk.bosch-connectivity.com/software-downloads
[7] www.cypress.com/documentation/development-kitsboards/s6sae101a00sa1002-solar-powered-iot-device-kit
[8] www.cypress.com/products/psoc-creator-integrated-design-environment-ide
[9] www.ti.com/ww/en/wireless_connectivity/sensortag2015/
[10] www.ti.com/tool/ccstudio-wcs

Elektor Presents: Rebecca Geier's Smart Marketing for Engineers **Workshop** Tour

With the proliferation of marketing channels and the buying process moving online, marketing planning has become a critical step to driving business success and ROI. Today, business and marketing leaders struggle with tough choices: How do I get my company found online? Do I go to the same trade show or downsize my booth and increase investment in SEO? Do I create more content and videos or advertise on LinkedIn? And the questions and trade-offs go on. Using a proven, research-based methodology for driving demand and growing sales pipelines, this workshop will help you answer these questions and more.


Smart Marketing For
ENGINEERS™
An Inbound Marketing Guide to Reaching Technical Audiences
REBECCA GEIER
Named one of the ten Most Innovative Entrepreneurs, Wall Street Journal

At the end of this **workshop**, you will have:

- A prioritized plan for marketing investments
- A draft campaign and content marketing plan
- Buyer personas defined
- Content themes and specific topics outlined
- Keywords identified
- A content repurposing plan to amplify your work
- Calls-to-action "paths" at each stage of the funnel to drive engagement and conversion
- Metrics to measure marketing ROI

About Rebecca Geier

Rebecca has over 25 years of global marketing experience in engineering and scientific markets. Named by The Wall Street Journal editors among the Ten Most Innovative Entrepreneurs in America, Rebecca released her debut book, *Smart Marketing for Engineers: An Inbound Marketing Guide to Reaching Technical Audiences in 2016*.

Prior to co-founding TREW Marketing, Rebecca managed marketing programs at a variety of technology and marketing companies and was a member of the marketing leadership team at National Instruments for 14 years. Rebecca is a regular speaker and writer on the subject of effectively marketing to technical audiences.

About the book

Rich in data showing engineers online content and browsing behaviour and preferences, the book is filled with lists, templates, and examples from B2B engineering and scientific companies in industries including automotive, military/aerospace, consumer electronics, manufacturing, IT, and oil/gas. If you are an engineer or technical business leader looking for a straightforward, research-based guide to modern marketing or a professional marketer new to targeting technical audiences, this book is for you.

WHAT:	One-and-a-half day marketing workshop
WHO:	Business and marketing leaders and teams who target highly technical audiences such as engineers or scientists
WHEN:	June 2017
WHERE:	Berlin, Munich, Aachen, Eindhoven

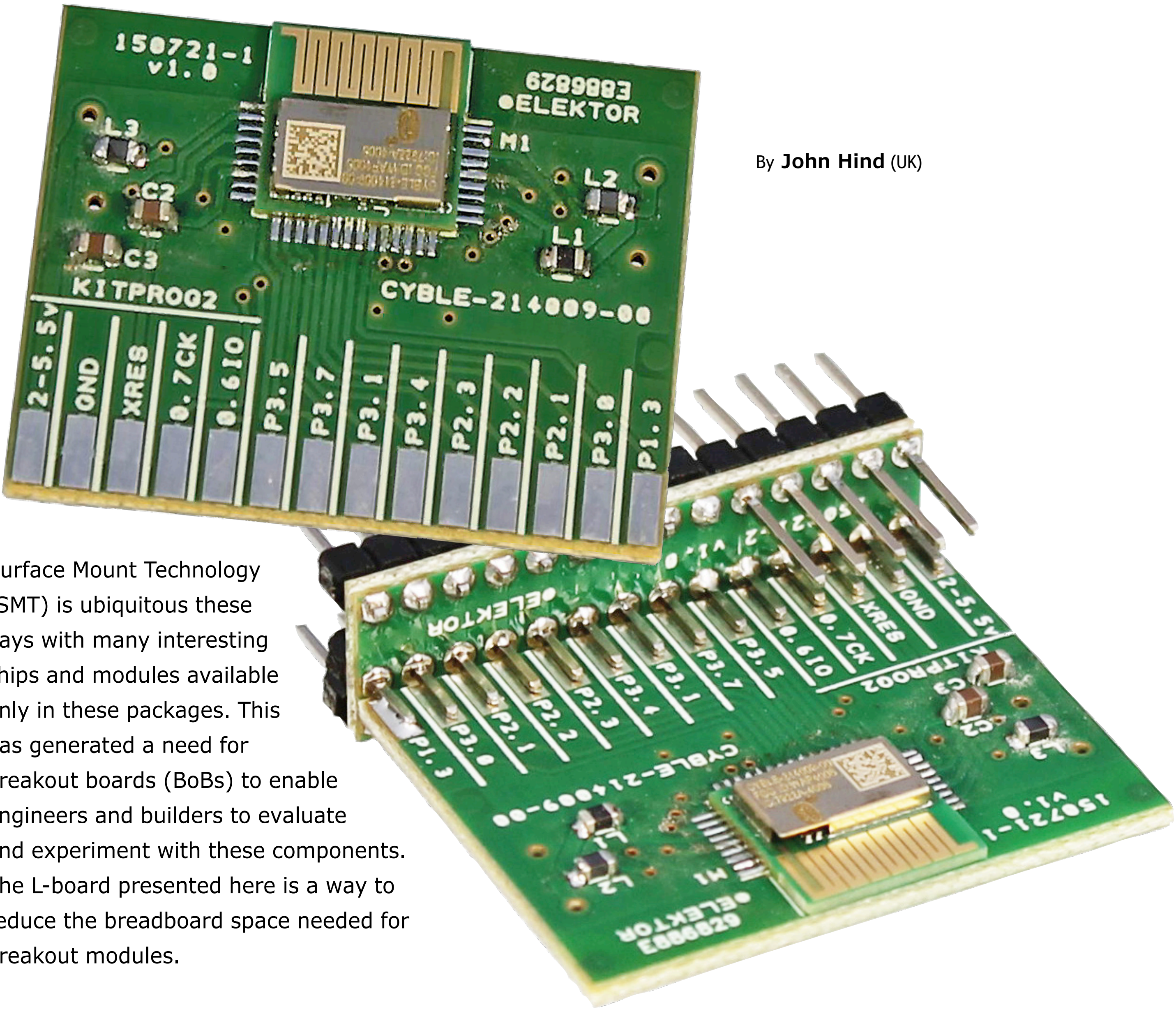
Further information:
www.elektormagazine.com/smart-marketing

LABS PROJECT

A PSoC BLE Module in a Breadboard-friendly L-board Format

for easy & comfy Bluetooth prototyping

By **John Hind** (UK)



Surface Mount Technology (SMT) is ubiquitous these days with many interesting chips and modules available only in these packages. This has generated a need for breakout boards (BoBs) to enable engineers and builders to evaluate and experiment with these components. The L-board presented here is a way to reduce the breadboard space needed for breakout modules.

Normally breakout boards fan the tiny SMT contacts out to the common and more human-scale 2.54 mm (0.1”) pitch used in pinheaders and breadboards. For devices with relatively low pin counts such as sensors and actuators, a single row of pins works well and can easily be fitted with a pinheader and plugged into a standard breadboard (a good example is the breakouts used in the current “Sensors Make Sense” articles series). However, as the pin count increases,

Features

- DIP–style, breadboard-space-saving module
- PSoC with Bluetooth Low Energy (BLE) 4.1
- Graphical programming interface

single in-line (SIL) breakouts become unreasonably long and are unstable on a breadboard. A dual in-line (DIL) structure improves this situation and many breakout boards use this format (for example, the various "T-board" projects published by Elektor over the years). But these bring their own problems when used on a standard breadboard with a central "gutter" designed for traditional DIP IC packages. If the board is more than 10 mm (0.4") wide, breadboard socket points will be occluded restricting the space for building peripheral circuits. A 10-mm board requires through-hole solder pads on each side leaving at best 6 mm (0.24") between the pads, which is inadequate space for most packages or even for routing more than a few PCB tracks off to the side as required by the T-board format. This fact invariably forces the board to be made wider than the 10-mm ideal.

Enter the L-board

The L-board format proposed here addresses these problems. It is a two-part board comprising a horizontal 10-mm-wide gutter board soldered to a vertical double-sided carrier board for the chip. This structure plugs into a standard breadboard, spanning the gutter without occluding any socket points. Multiple L-boards may be placed side by side on the breadboard just like the DIP packages of old. The penalty is that, like the famous Swedish furniture, the flat-pack L-board requires some assembly before it is ready for use.

My First L-board

Cypress Semiconductors PSoC BLE modules make for an interesting implementation of the L-board concept. PSoC stands for Programmable System on Chip and in this case it combines a powerful ARM 32-bit processor with a Bluetooth Low Energy (BLE) radio peripheral, programmable digital logic and analog peripherals with internal switching fabric. It is worth mentioning that there is also a Programmable Radio on Chip (PRoC) family. The PSoC BLE chips are actually a superset of the PRoC functionality with the later omitting the programmable digital peripheral. But they are broadly interchangeable from a development point of view.

As the 'C' in the name implies, these components are available in chip form, but the alternative micro-modules that we will use here have an integrated antenna assembly and oscillator crystals and are prequalified, greatly simplifying low volume manufacture. Unfortunately, the modules are almost impossible to solder without professional SMT assembly equipment.

The PSoC/PRoC range has a particularly low startup cost for developers due to subsidized development modules and a free-to-download Integrated Development Environment (IDE). To get started,

PROJECT INFORMATION

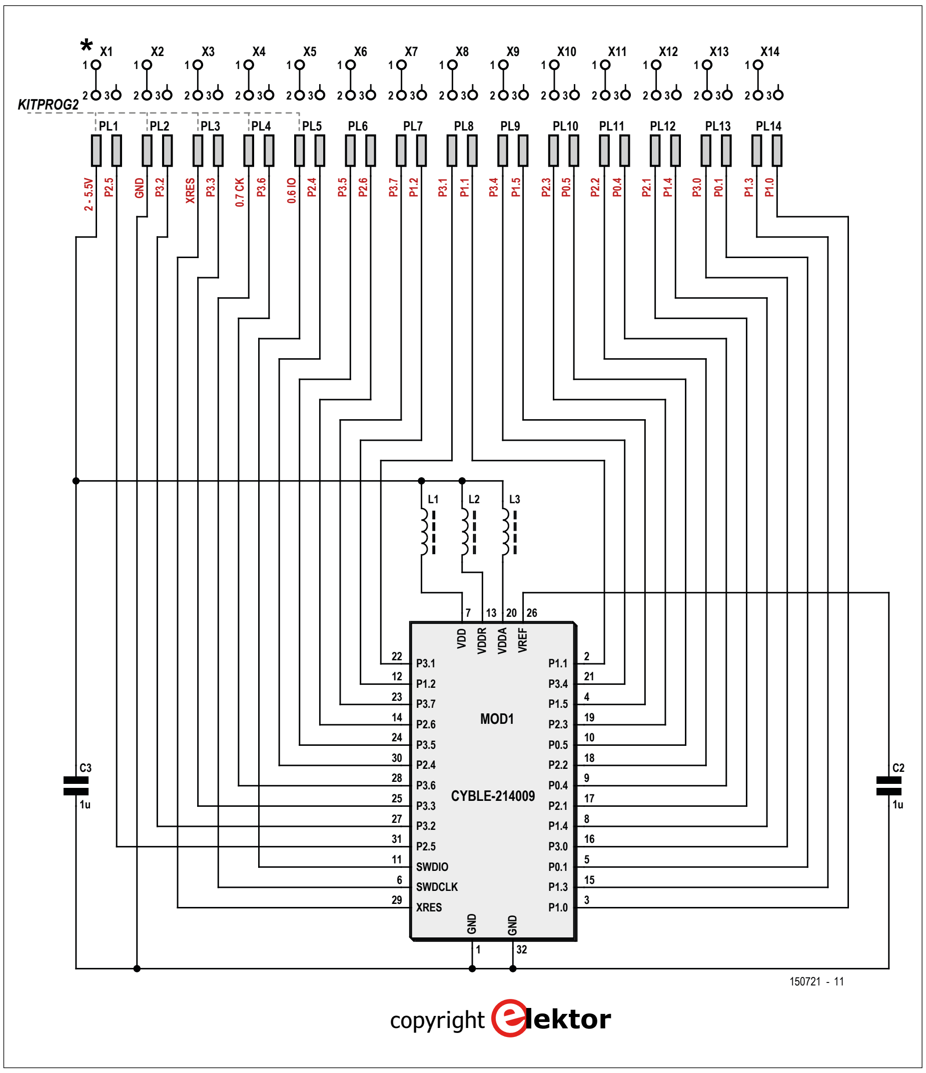
T-board L-board PSoC BLE Prototyping BoB

entry level
➡ intermediate level
expert level

2 hours approx.

Soldering iron, Breadboard, Kitprog2 programmer, Windows PC, PSoC Creator

€50 / £45 / $55 approx.



X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14
KITPROG2
PL1 PL2 PL3 PL4 PL5 PL6 PL7 PL8 PL9 PL10 PL11 PL12 PL13 PL14
2 - 5.5V P2.5 GND P3.2 XRES P3.3 0.7 CK P3.6 0.6 IO P2.4 P3.5 P2.6 P3.7 P1.2 P3.1 P1.1 P3.4 P1.5 P2.3 P0.5 P2.2 P0.4 P2.1 P1.4 P3.0 P0.1 P1.3 P1.0
L1 L2 L3
VDD VDDR VDDA VREF
MOD1
CYBLE-214009
P3.1 P1.2 P3.7 P2.6 P3.5 P2.4 P3.6 P3.3 P3.2 P2.5 SWDIO SWDCLK XRES
P1.1 P3.4 P1.5 P2.3 P0.5 P2.2 P0.4 P2.1 P1.4 P3.0 P0.1 P1.3 P1.0
GND GND
C3 1u
C2 1u
150721 - 11
copyright elektor

Figure 1. Schematic of the L-board for the CYBLE-214009-00 PSoC module with support for Bluetooth LE 4.1. PL1 to PL14 represent the edge connector of the main board, X1 to X14 represent the three pinheader rows on the 'gutter' board.

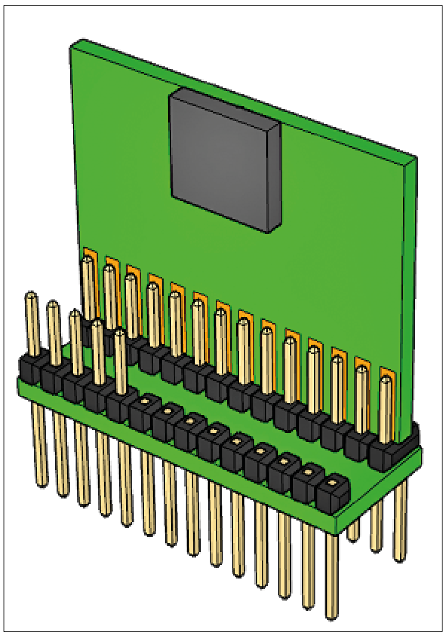


Figure 2. 3D rendering of the L-board. Not visible are the pins that connect the backside of the vertical board to the horizontal board.

it is only required to buy one of several prototyping kits featuring the Kitprog2 USB programming and debugging adapter, some of which cost under €/$/£10 (watch out however for the very similar kits which feature only a USB bootloader adapter rather than the full Kitprog2).

Figure 1 presents the schematic of our (your!) first L-board. Since it is a breakout board, not many components are used. All we need is the module, some decoupling capacitors and a few ferrites to keep the noise down. And of course pinheaders.

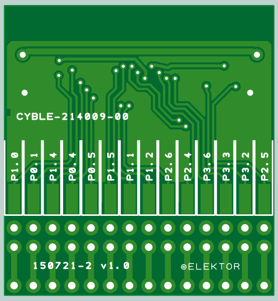
It would have been ideal to use the extended-range, BLE4.2-supporting CYBLE-224116-01 module which is currently sampling, but it was not ready in time for this article so the BLE4.1 CYBLE-214009-00 was used instead. Hopefully the design can be upgraded soon since the extended range will open BLE to a broader application space beyond the original cable replacement or personal area network (PAN). A PCB design for this newer module is already available on the Labs page for this article [1].

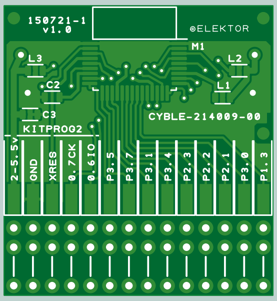
L-board assembly

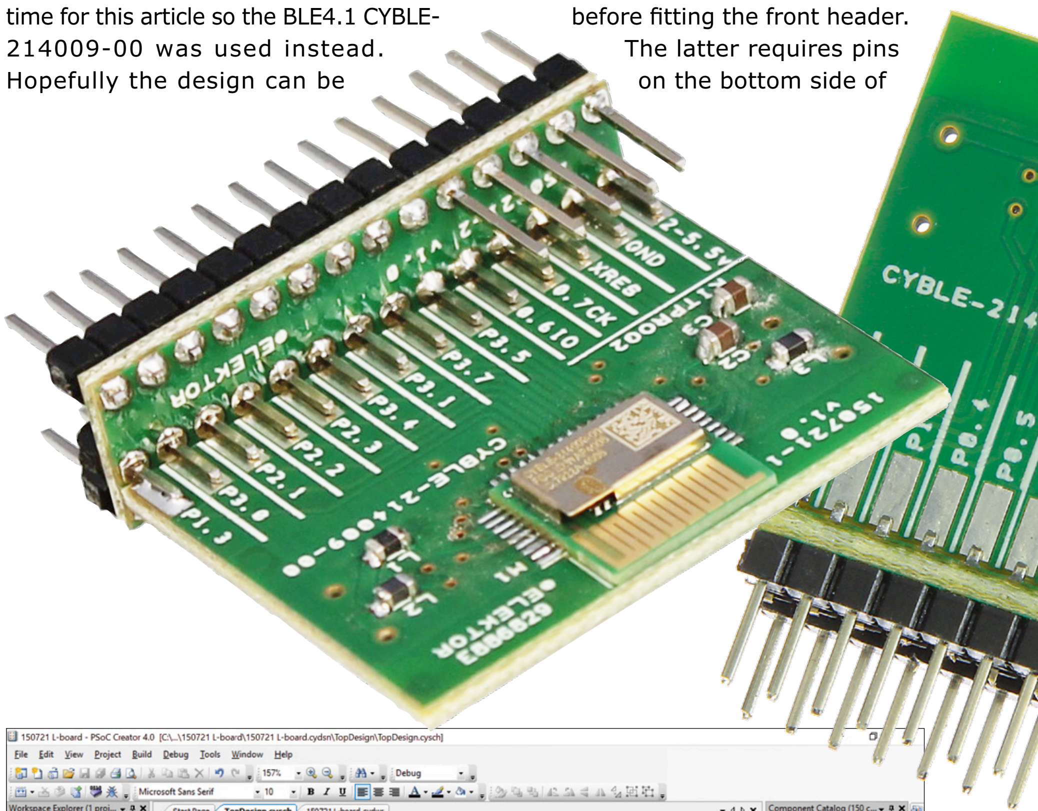
The L-board design uses a so-called V-cut between the two board parts allowing them to be snapped cleanly apart. Final assembly requires manually soldering three 14-pin pinheaders (for this 28-pin implementation) to the gutter board. The rear header requires pins at the top and bottom of the board. The middle header requires pins on the topside only, so the bottom pins should be cut off after soldering. This is much easier if done before fitting the front header. The latter requires pins on the bottom side of

COMPONENT LIST

C1,C2 = 1µF, 0805
L1,L2,L3 = ferrite, 300Ω @ 100MHz, 0805
MOD1 = CYBLE-214009-00
2x 14-pin pinheader, 0.1" pitch, ~15mm pin length
1x 14-pin pinheader, 0.1" pitch, ~7mm pin length
PCB #150721-1


CYBLE-214009-00
P1.0 P0.1 P1.4 P0.4 P0.5 P1.5 P1.1 P1.2 P2.6 P2.4 P3.6 P3.3 P3.2 P2.5
150721-2 v1.0
©ELEKTOR


150721-1 v1.0
©ELEKTOR
M1
L3
L2
C2
L1
C3
KITPROG2
CYBLE-214009-00
2-5.5V GND XRES 0.7CK 0.6IO P3.5 P3.7 P3.1 P3.4 P2.3 P2.2 P2.1 P3.0 P1.3



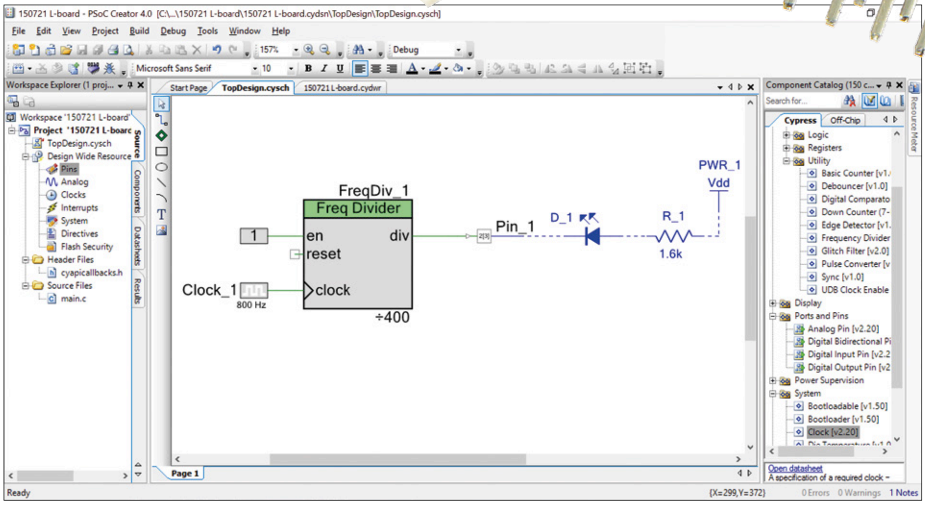
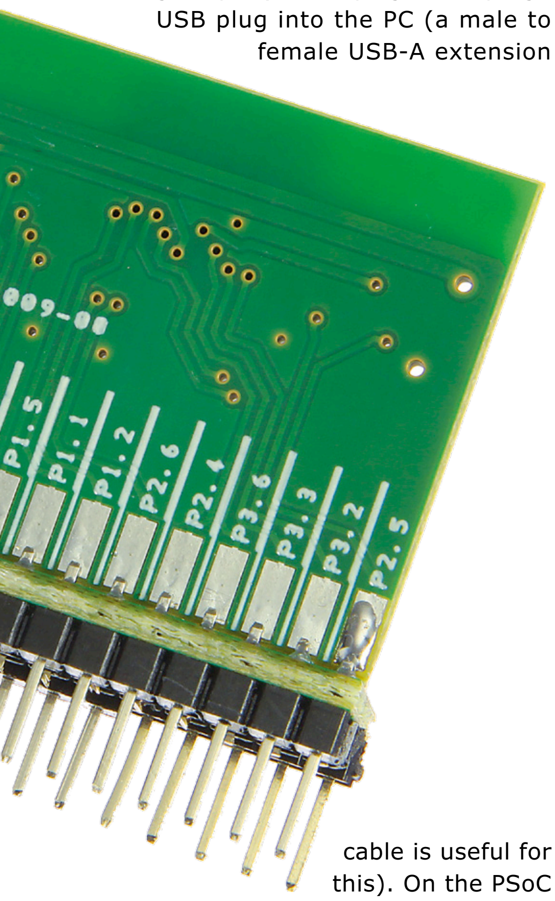


Figure 3. "Hello, world!" programmed this way in PSoC Creator does not use the microcontroller at all. Notice that everything in blue must be added on the breadboard.

the board, but, in this case, the leftmost five pins should be left on the top side as well because these form the connector for the Kitprog2 adapter. Finally, the main board part is positioned between the first and second row of pins on the top of the gutter board which are then soldered directly to the pads on each side of the board (**Figure 2**). The assembly can now be mounted across the gutter of a breadboard.

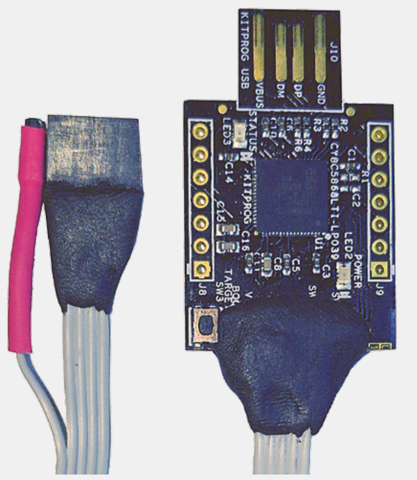
Hello, world!

To get started with programming, download the PSoC Creator IDE from the Cypress web site and install it on a PC (it is Windows only, unfortunately). Connect the Kitprog2 ribbon to the pins at the right-hand end of the gutter board (see inset on how to make a programming adaptor). Then plug the Kitprog2 USB plug into the PC (a male to female USB-A extension cable is useful for this). On the PSoC Creator 'Start Page', select 'Create New Project'. Select as 'Target module' the CYBLE-214009-00. On the next page, select 'Empty schematic' and on the final page select an appropriate folder to store the files. It may be pleasantly surprising at this point to be looking at a blank schematic page rather than the more familiar code editor! To the right of this is the 'Component Catalog'. Drag a 'Digital Output Pin' from the 'Ports and Pins' section onto the schematic, right-click on it and select 'Configure'. On the 'General' tab check the boxes 'Digital output', 'HW connection' and 'External terminal'; click 'OK' when done. Add an LED, a resistor and a power component from the 'Off-Chip' tab in the 'Component Catalog' and arrange and connect them as shown in **Figure 3**. In the workspace tree to the left of the schematic, double-click on 'Pins' under 'Design Wide Resources'. You will see a picture of the module pin-out and a list of the pins on the schematic, consisting for the moment of the one output pin added earlier. In the 'Port' drop-down assign this to a physical port. Any one will do, but pick 'P2[3]' for now. This is labeled P2.3 on the L-board. Of course, the physical LED circuit has to be built on the breadboard (the off-chip components on the schematic are for documentation only).



Programming adapter

To make a programming adapter, cut the Kitprog2 portion off a prototyping kit (which is break-routed to facilitate this) and solder a length of 5-core ribbon cable to the five pin holes at the opposite end to the USB plug. Solder the other end of the ribbon cable to a 5-pin single-row pin socket. It is best if this is separated into a one-pin section for VTARG and a four-pin section for the other signals. This allows the board to be either powered from the USB connection to the Kitprog2 or alternatively separately powered from the breadboard pin (a unique benefit of the PSoC architecture is an unusually wide operating voltage range stretching all the way from 1.9 V to 5.5 V).



It is also possible to solder a five-pinheader to the original target board so that it can also be used with the separated Kitprog2. The photograph shows a Kitprog2 assembly, finished using a moldable rubber product called Sugru which is excellent for making ad hoc back shells for connectors and other insulated electromechanical assemblies in prototyping.

Click the 'TopDesign.cysch' tab to return to the schematic. From the 'Cypress' tab of the 'Component Catalog', add a 'Clock' component (to be found in the 'System' folder) and set its properties to 800 Hz. This is close to the minimum frequency available from the built-in clock dividers.

Next add a 'Frequency Divider' component (it is in the 'Digital\Utility' folder) and set it to divide by 400, connecting its clock terminal to the clock and its 'div' output to the pin. Finally, add a 'Logic

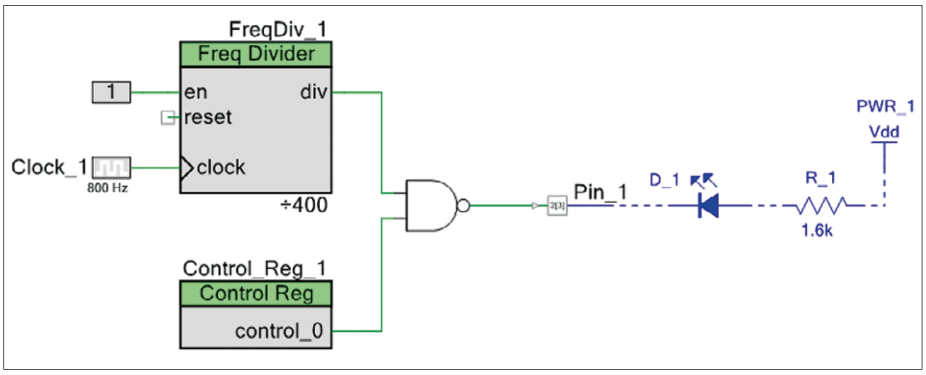


Figure 4. In the extended "Hello, world!" example the LED is controlled by both the microcontroller and the programmable digital logic blocks.

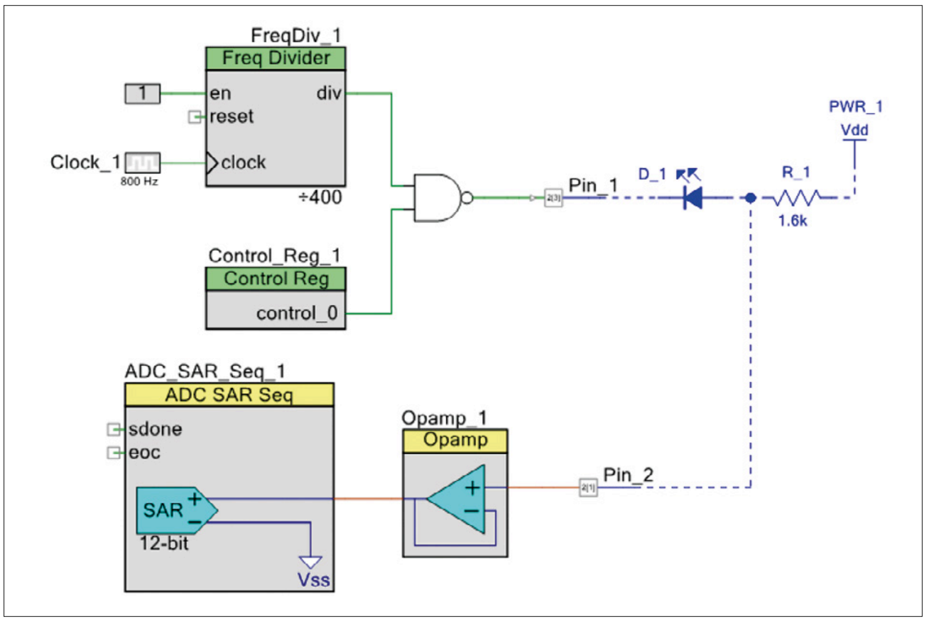


Figure 5. Adding an ADC block to the circuit allows measuring the forward voltage of the LED.

Listing 1. A simple program to try out debugging and stepping through your code.

```
#include “project.h”

int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    for(;;)
    {
        Control_Reg_1_Write(0x01);
        Control_Reg_1_Write(0x00);
    }
}
```

Listing 2. The program that completes the circuit of Figure 5.

```
#include “project.h”

int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    volatile float32 volts = 0;
    ADC_SAR_Seq_1_Start();
    ADC_SAR_Seq_1_StartConvert();

    for(;;)
    {
        Control_Reg_1_Write(0x01);
        volts = ADC_SAR_Seq_1_CountsTo_Volts(0,
ADC_SAR_Seq_1_GetResult16(0));
        Control_Reg_1_Write(0x00);
    }
}
```

High’ (see ‘Digital\Logic’ folder) and connect it to the ‘Enable’ pin on the divider. You should now have something similar to figure 3. Save your work.

Now take the ‘Program’ option on the ‘Debug’ menu. After the design is built and downloaded, the LED should immediately start flashing. This is the traditional embedded equivalent of “Hello, world!” implemented without writing a single line of code. In fact, the ARM processor is not even running: this works entirely in programmable logic!

Add a ‘Control Register’ and a ‘Nand’ gate to the schematic and set the former’s number of ‘Outputs’ to 1. Make the connections as shown in **Figure 4**. In the ‘Workspace Explorer’ on the left, open the ‘main.c’ file and add two lines of code inside the ‘for’ loop (see **Listing 1**):

```
Control_Reg_1_Write(0x01);
Control_Reg_1_Write(0x00);
```

This time select ‘Debug’ on the ‘Debug’ menu. This starts the ARM processor and halts it before the first code line. Press F10 twice and notice that the LED starts flashing. Keep pressing F10 and notice that the LED flashes when the control register is set and remains off when it is reset.

Measuring a voltage

To explore the analog capabilities, let’s measure the forward voltage of the LED. First go to the ‘System’ page of the ‘Design Wide Resources’ and set the supply voltages to 5.0 V (assuming the board is powered from the Kitprog2). Add an ‘Analog’ pin, an ‘Opamp’ and a ‘Sequencing SAR ADC’ to the schematic. Configure the pin to have an ‘External terminal’, the opamp to ‘Follower’ and the ADC to a sampling rate of 100,000 samples per second. Set its ‘Vref select’ to ‘VDDA’, its ‘Sequenced channels’ to 1, enable channel 0 on the ‘Channels’ tab and set its ‘Mode’ to ‘Single’. On the ‘Pins’ tab assign ‘P2[1]’ to the pin. Wire up as shown in **Figure 5**, remembering to add the wire between the LED and the analog pin physically on the breadboard as well! Now add a few more lines of code to Listing 1 to obtain **Listing 2**:

```
volatile float32 volts = 0;
ADC_SAR_Seq_1_Start();
ADC_SAR_Seq_1_StartConvert();
```

In the 'for' block, between the two control register write instructions we added earlier, add:

```
volts = ADC_SAR_Seq_1_
CountsTo_Volts(0,
ADC_SAR_Seq_1_GetResult16(0));
```

Debug this and set a watch on the 'volts' variable. It could be anywhere between 1.8 and 4.5 volts depending on the color and type of LED, or it could be 0 volts if the reading is taken when the LED is in the off portion of the flash cycle. Take a look at the 'Analog' tab of 'Design Wide Resources' to see how the analog schematic has been implemented in the chip. It is also worth checking the 'Resource Meter' accessed via a tab on the top right edge of the application. You may be alarmed that making an LED flash has consumed nearly half the available UDB resources, but as always there is more than one way to skin a cat. One of the available four 'Timer Counter PWM' resources could have been used, or the traditional method of using a timer interrupt and code is also possible.

Adding Bluetooth to the mix

Bluetooth Low Energy is covered in a downloadable example project [1], but let's have a quick look. Drag a 'Bluetooth Low Energy' component from the 'Communications' section of the catalog and open its configuration dialog. It is possible to implement one of the many approved standard profiles, or set profile to 'Custom' to design a new profile. Usually this will be a GATT Server and GAP peripheral (these are the types which can be connected to smartphones, tablets or PCs). The characteristics and descriptors are then defined in the 'Custom Service' node on the 'Profiles' page. Notice, however, that this module can also implement the central role and even dynamically switch between roles. This potentially allows point-to-point links or even networks to be implemented using multiple modules with no need for a smartphone, tablet or PC. The download provides more details and some generic code for implementing a custom service as well as a fully worked example.

It is also worth checking out Cypress's EZ-Serial firmware, particularly if not comfortable with programming in C. The free CySmart app for Android, iOS and Windows is an excellent tool for debugging and testing BLE devices (although the Windows version does require purchase of a custom USB dongle).

Conclusion

This article introduced the L-board, a new format for breakout boards which has some significant advantages over the established T-board format and other "flat" breakouts. A sophisticated BLE PSoC module was implemented in L-board format and some experiments introducing the advanced features of this module were described. Cypress is currently sampling a later generation of these modules supporting BLE 4.2 with improved speed and capacity, better security and, crucially, enhanced range. It should be possible to follow up with one of these modules in L-board format once they are shipping. The enhanced range will open BLE up to exciting new application areas beyond the current cable replacement or personal area network duties. These include home automation and possibly a new generation of low-cost mix-and-match instrumentation for both laboratory and field work. ⏮

(150721)

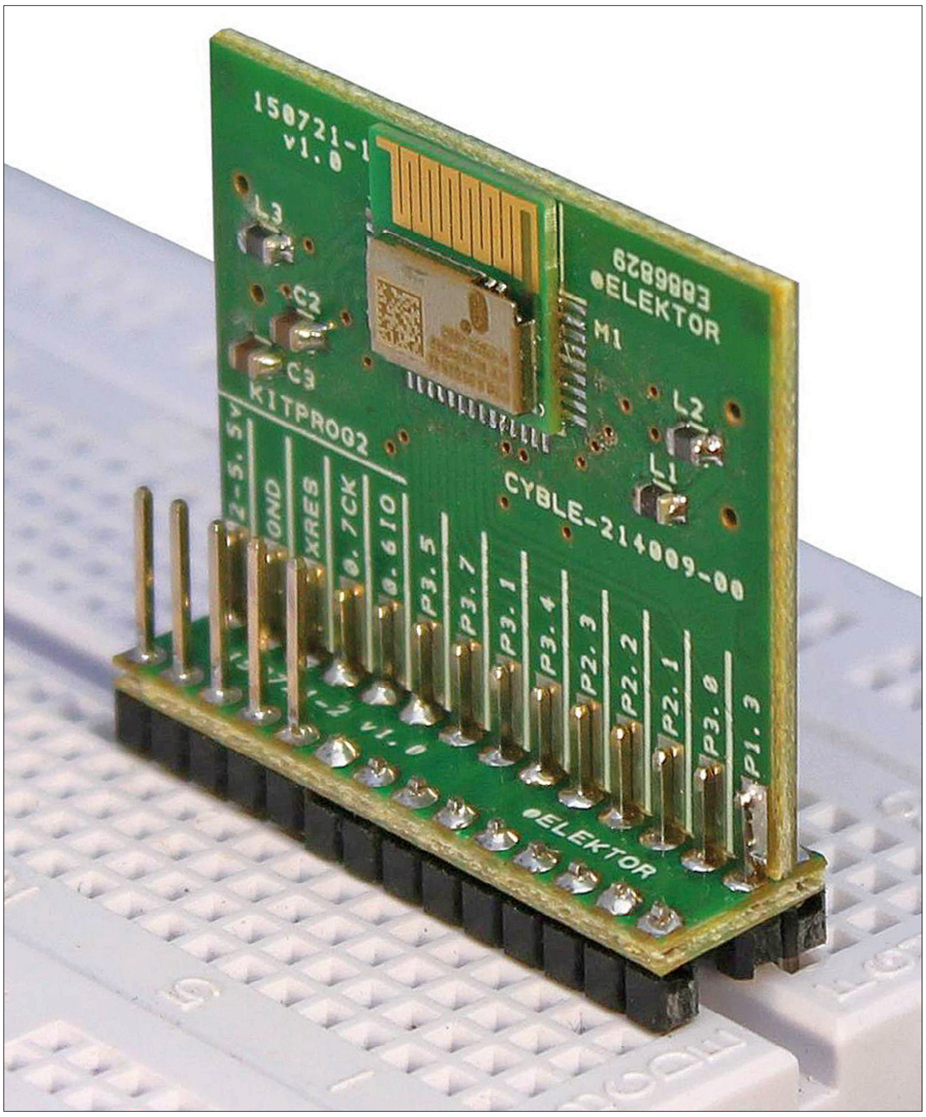


Figure 6. Assembled L-board prototype plugged into a breadboard.

Web Links

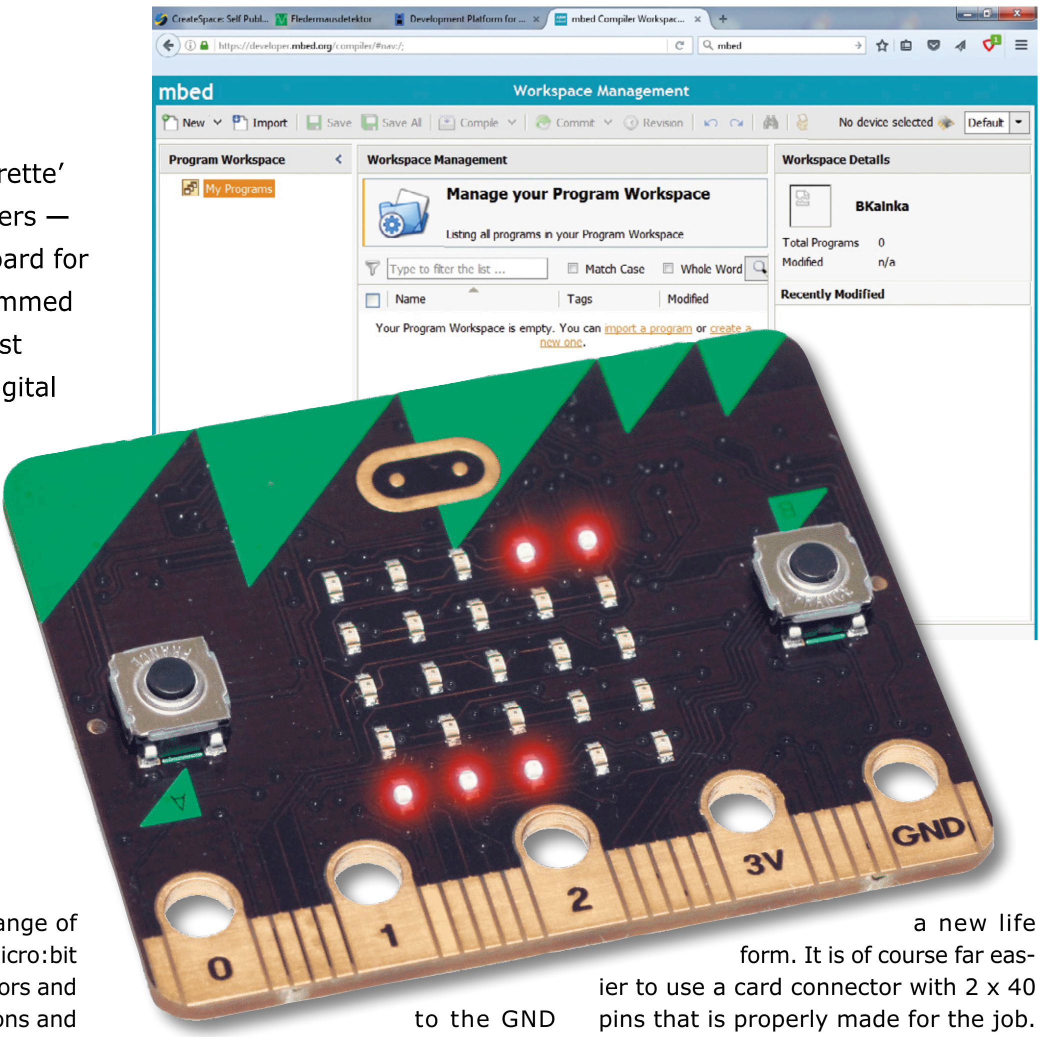
[1] www.elektormagazine.com\labs\l-board

[2] www.elektormagazine.com\150721

BBC micro:bit for Electronicists
In bed with mbed

By **Burkhard Kainka** (Germany)

The BBC micro:bit 'computerette' is not only for school-age users — it's also a great controller board for grown up electronicists. Crammed into its tiny footprint is almost everything that you need: digital inputs/outputs with PWM and A-D converters along with a USB interface for power supply, programming and transferring data. A variety of sensors and Bluetooth round off the configuration.



Your Program Workspace is empty. You can import a program or create a new one.

Sure, an Arduino also offers a range of inputs and outputs but the BBC micro:bit offers additionally numerous sensors and control functions. Two input buttons and a 5x5-LED display field, a compass sensor and a 3-axis accelerometer, plus light and temperature measurement. As the crowning glory you have Bluetooth Low Energy (BLE) data transfer.

Vital connections

The PCB was developed by the BBC in collaboration with the University of Lancaster as a teaching aid for schools. For this reason it is equipped with five oversized connection pads with 4 mm holes, to which crocodile clips can be linked easily (**Figure 1**). In theory these clips might also touch the smaller pins on either side but the designers took this risk into consideration. The pins adjacent to the GND and 3-V connections carry the same potential, and next to the large Port connections 0, 1 and 2 you will also find Port pins that will survive any direct cross-connection without complaining.

You can even use croc-clip cables for your first simple experiments (see header photo). When things get more complex and you need more connections, you'll require some kind of connector plugs if you wish to avoid soldering wires. With its 0.05- mm pitch connections, the board happens to match the card slots of a discarded PC motherboard, so with a pair of pliers, small saw and soldering iron you could well upcycle this into a new life form. It is of course far easier to use a card connector with 2 x 40 pins that is properly made for the job. Custom-made boards are available in the Elektor Store [6][7]. **Figure 2** shows the PCB for the BBC micro:bit Weather Station project.

Alternatively you could use dual-row header strips with 2 x 20 pins that you simply solder to the gold contacts (**Figure 3**). The pins of the header strips are spaced at 2.54 mm, which is twice the pitch of the small contacts on the PCB. The workaround is to solder pins to every second contact on the BBC micro:bit. On the rear side of the board things are much the same but you need to take care that all connections on the back are totally isolated. The pins soldered on that side serve primarily only for mechanical stability. The rear-side pins do have a secondary purpose, however: if you need to make use of a signal on the micro:bit that you have not employed so far, you can connect this to one of the rear-side pins with a short piece of wire. If you intend to plug the Micro:bit into another PCB, then you will need to remove the corresponding pin in the front-side row to avoid creating a short circuit. There are of course plenty of pins that are not required, for instance the auxiliary connections for GND and VCC. What might be critical on the other hand are the I²C connections, not all of which can be handled using the header strip method. To resolve this we remove the superfluous pin at the GND connection. This is very simple: heat the pin with a hot soldering iron tip, then pull it out with pliers. You can now poke some wire through the opening previously occupied by the pin and then solder one end to the desired connection point (SDA, P20). On the rear side you solder the wire to the corresponding pin, ensuring the integrity of the I2C bus is preserved.

Our first programs

Practical applications in the realm of electronics are really simple to fix up. Several programming languages that were designed primarily for educational use in schools [1][5] are suitable. Here, however, we will work with C++, in which practically everything is possible, whereas other languages always have certain limitations. Using the mbed platform [2] is particularly straightforward, because you can work online without having to install anything.

Log into mbed and set up an area of your own, if you have not already done this for other projects. If you now attempt to load an existing example, you will be reminded to select a platform, in other words a system with which you wish to work. You will be referred to the Hardware page and here you may be amazed at how many boards are already usable, one being from Elektor by the way. In this situation the micro:bit is the 'target system'. Here you will come across further information, such as the circuit diagram of the Micro:bit and a button to 'add to your mbed Compiler'. There are also crucial links to the documentation of Lancaster University and some entry-level sample code.

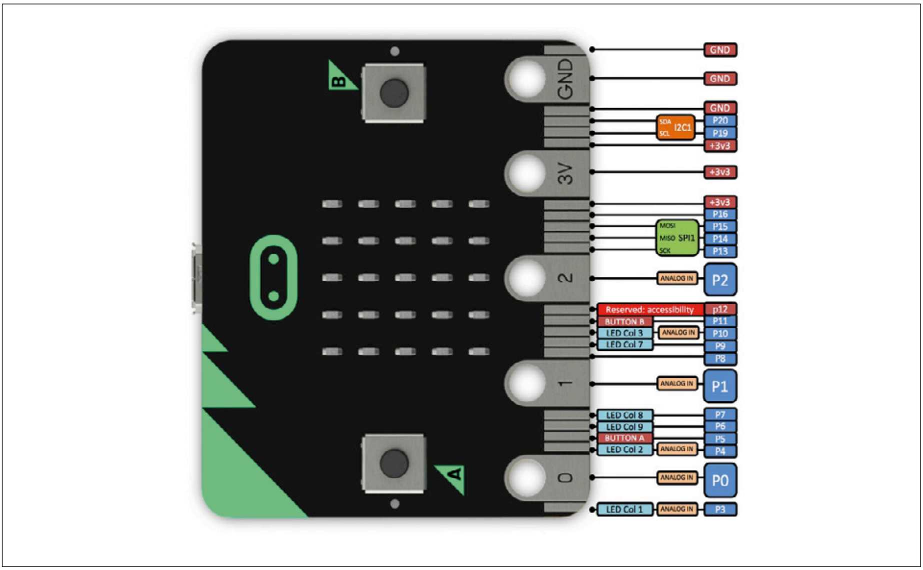


Figure 1. Connections of the BBC micro:bit.

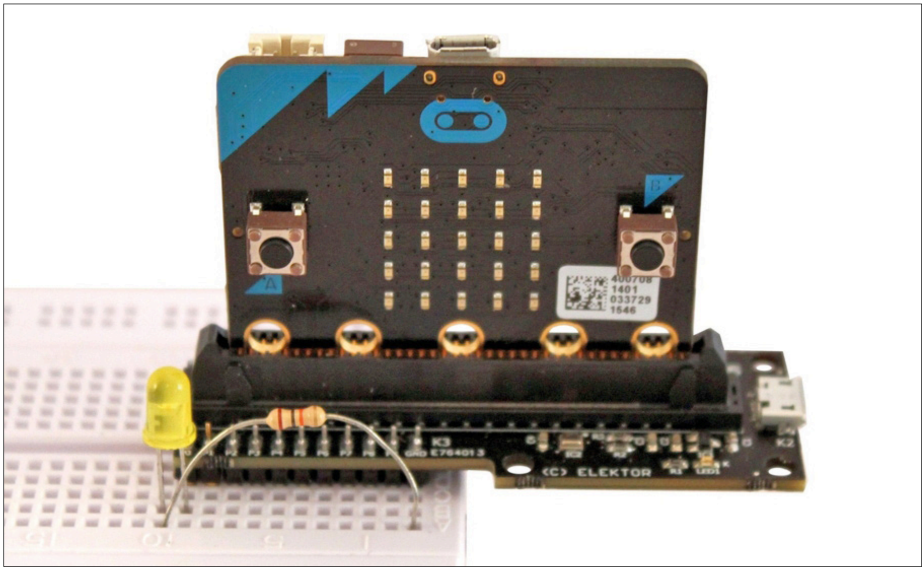


Figure 2. Elektor PCB 150652-1 used as a breadboard.

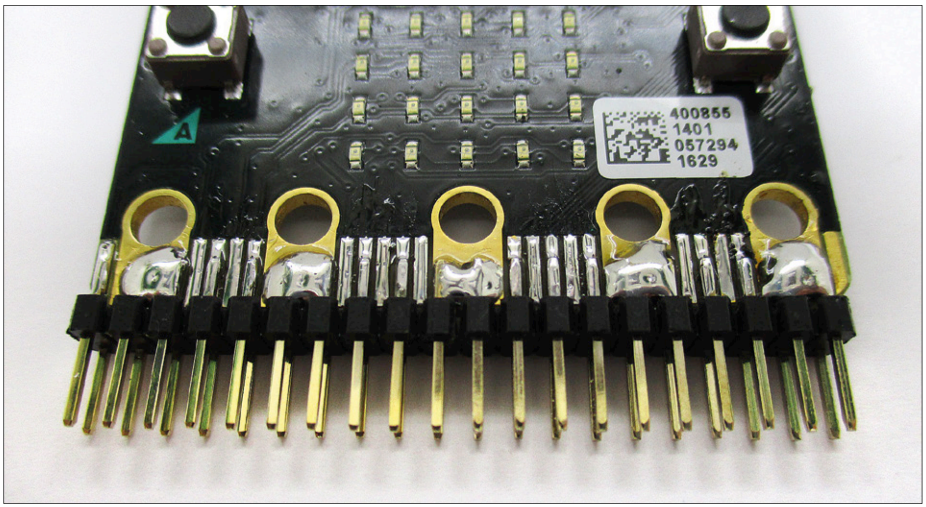
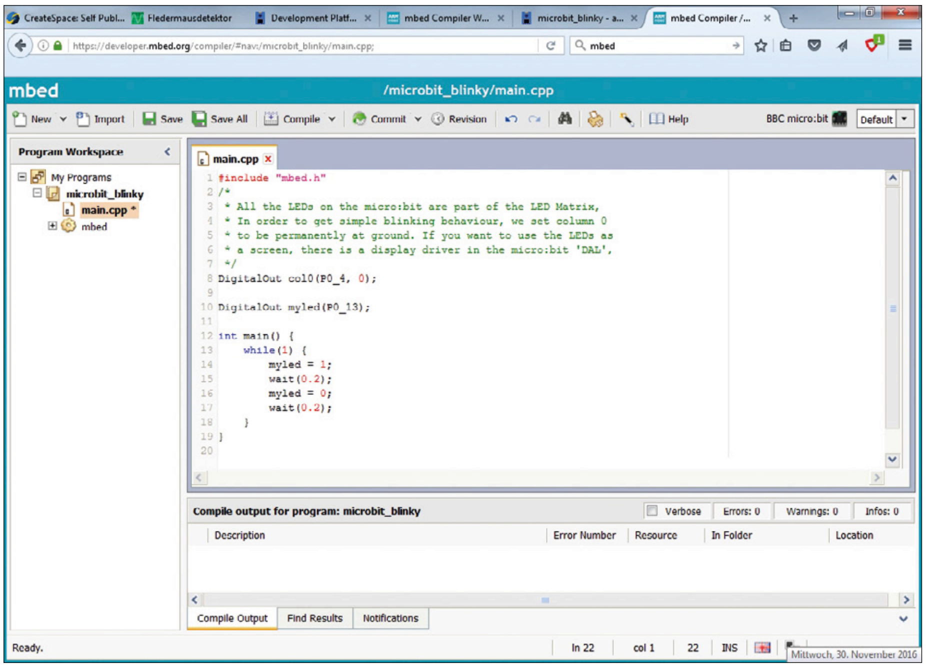
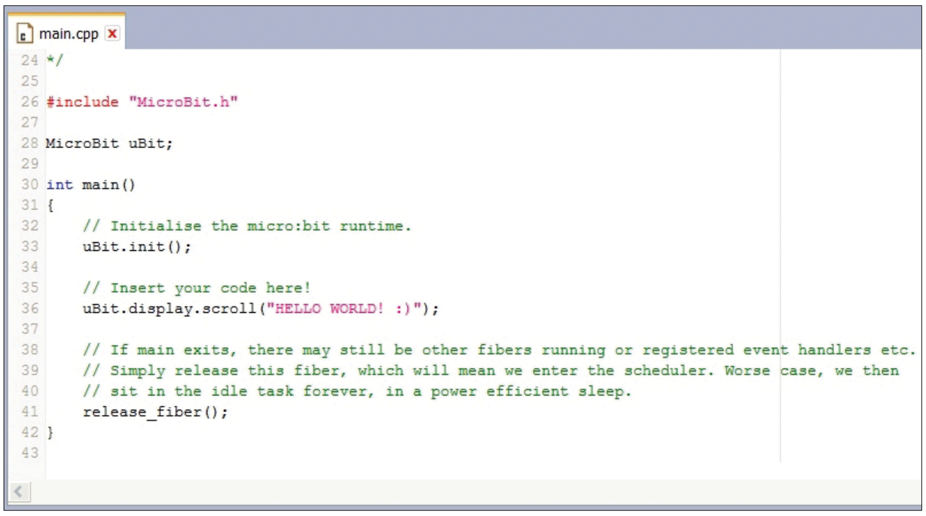


Figure 3. Header pin assignments.



```
#include "mbed.h"
/*
 * All the LEDs on the micro:bit are part of the LED Matrix,
 * In order to get simple blinking behaviour, we set column 0
 * to be permanently at ground. If you want to use the LEDs as
 * a screen, there is a display driver in the micro:bit 'DAL',
 */
DigitalOut col0(P0_4, 0);

DigitalOut myled(P0_13);

int main() {
    while(1) {
        myled = 1;
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
}
```

Figure 4. The micro:bit Blinky project.



```
*/

#include "MicroBit.h"

MicroBit uBit;

int main()
{
    // Initialise the micro:bit runtime.
    uBit.init();

    // Insert your code here!
    uBit.display.scroll("HELLO WORLD! :)");

    // If main exits, there may still be other fibers running or registered event handlers etc.
    // Simply release this fiber, which will mean we enter the scheduler. Worse case, we then
    // sit in the idle task forever, in a power efficient sleep.
    release_fiber();
}
```

Figure 5. The micro:bit Hello World project.

Listing 1. Voltage measurement.

```
//Voltage1
#include "MicroBit.h"

MicroBit uBit;

int main()
{
   uBit.init();
   MicroBitSerial serial(USBTX, USBRX);
   while (1) {
        int u = 3300 * uBit.io.P0.getAnalogValue()/ 1023;
        uBit.display.scroll(u);
        uBit.serial.printf("%d\r\n", u);
        uBit.sleep(500);
        }
}
```

The first of these is called *Micro:bit_ Blinky* and should be downloaded now. The compiler shows the newly installed platform and the imported program. Clicking on *main.cpp* opens the uncluttered source code (**Figure 4**).

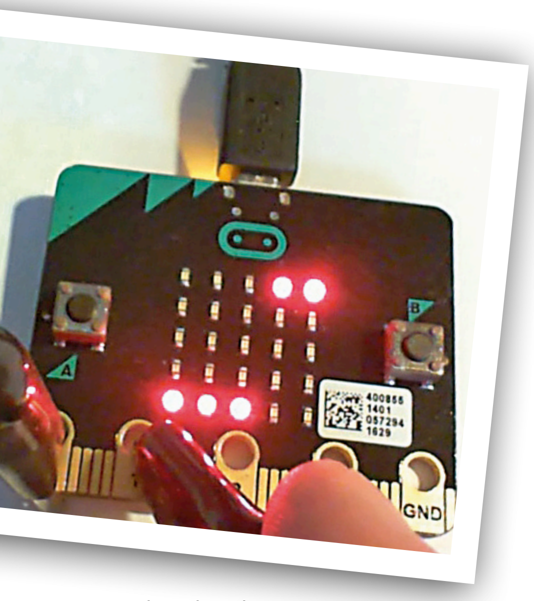
Something that will strike you immediately is the way that the Port connections are labeled P0_4 and P0_13, which conflicts with the official pin assignment. This is because when you install *mbed.h*, you are addressing the controller on the micro:bit but not the entire board with all its features and options. We can test the program now. Using *Compile* creates a hex file, which is then simply copied into the USB memory of the Micro:bit. The yellow status LED on the board begins to flash straightaway, indicating that the system is ready to start programming the controller. After one second the process is complete and you can see the result: the upper left-hand LED on the 5x5 LED display flashes.

The second example is called *Micro:bit Hello World* and embeds *microbit.h* (**Figure 5**). This makes it possible to employ the highly specific options that the board offers. Importing the program takes a noticeably long time, which indicates that very many files are being loaded. This is the great advantage of Mbed: everything you previously had to merge and integrate with great care Mbed now handles itself. With a different IDE on your own computer you could of course do this yourself, but you might well spend a long time doing this or even give up in desperation. With mbed you can work without even a moment's thought for what's going on in the background. In the example some scrolling script is produced. The mission-critical line is self-explanatory: uBit.display.scroll ("HELLO WORLD! :)"; compile, transfer, run.

You may be thinking, "So much effort for just a scrolling display?" But in reality Mbed offers much more, namely total access to all the vital hardware elements of the system, not only for display functions but also to Ports having A-D converters and PWM, to specialized sensors like compass and accelerometer functions and much more. In the Workspace you will find the *microbit* folder and

Listing 2. Displaying sensor values.

```
//Sensors
#include "MicroBit.h"
MicroBit uBit;
int main()
{
   uBit.init();
   MicroBitSerial serial(USBTX, USBRX);
   while (1) {
        uBit.serial.printf("Time: %d ms \r\n", uBit.systemTime());
        uBit.serial.printf("Temp: %d deg \r\n", uBit.thermometer.getTemperature());
        uBit.serial.printf("P0: %d mV  \r\n", uBit.io.P0.getAnalogValue());
        uBit.serial.printf("P1: %d mV  \r\n", uBit.io.P1.getAnalogValue());
        uBit.serial.printf("P2: %d mV  \r\n", uBit.io.P2.getAnalogValue());
        uBit.serial.printf("X: %d mG  \r\n", uBit.accelerometer.getX());
        uBit.serial.printf("Y: %d mG  \r\n", uBit.accelerometer.getY());
        uBit.serial.printf("Z: %d mG  \r\n", uBit.accelerometer.getZ());
        uBit.serial.printf("B: %d µT  \r\n", uBit.compass.getFieldStrength());
        uBit.serial.printf("\r\n");
        uBit.sleep(1000);
        }
}
```



within this the *microbit-dal* (Device Abstraction Layer) sub-folder with an unbelievable number of files through which you can rummage for hours in order to discover all the options.
A quicker way would be to read Lancaster University's own documentation [3]. In this you can find all the important information on every topic, also snippets of code that you can copy and insert into your own source code. You could easily expand the existing source code *main.cpp* of the micro:bit Hello World project too and try out everything, one item after another. An alternative would be to clone the project and then continue working on the copy. The cloned project will now be called, for instance, *microbit-test* and will work the same as the original initially. Following on from this, we have presented some brief source code samples that can be copied into *main.cpp* easily and then tested in this way. All of these programs can be downloaded in .txt format from the Elektor website [8].

Measuring voltages

What we all need constantly in our personal electronics labs are A-D inputs. The micro:bit has a 10-Bit A-D converter with six potential inputs, three of which are taken out to the large connection pads. For testing its characteristics we have written a simple program (**Listing 1**). Using *uBit.io.P0.getAnalogValue()* you obtain a 10-bit value for the voltage range up to 3.3 V. This is converted into mV and then displayed.

The micro:bit's typical method of displaying data is scrolling text on the LED display. This is highly functional but the legend displayed does require a fair degree of attentiveness to read it. Fortunately there is also a serial interface enabling you to make the data visible on a terminal screen. To use all this under Windows we need to install the mbed Windows Serial Port driver. How you do this is described in [4].

If you connect Input 0 to the 3-V connection on the board 3300 mV is displayed, as expected. Comparative measurements prove that the actual voltage is slightly smaller. That's because the board stabilizes the 5 V supply from the USB input to 3.3 V using a voltage regulator followed by a Schottky diode to produce V_{CC}.
With an open input the measurement indicates about 880 mV. If you measure the voltage simultaneously using a high-impedance oscilloscope, you will see a higher voltage that nosedives (collapses) shortly after each measurement. In point of fact the three most significant connections 0, 1 and 2 are each provided with pull-up resistors of 10 MΩ, enabling the digital Port pins to serve as touch sensors without extra effort. The true open-circuit voltage should thus lie close to 3.3 V. To measure this as well you can connect a 100 nF capacitor to the input. This will then charge itself up to the open circuit voltage and buffer this during the measurement.

Sensors galore

Our second little sample program (**Listing 2**) indicates numerous different measurement readings from the various inputs and sensors. Among these are the temperature of the microcontroller and the system time in milliseconds since the last start. You also have the three voltages on Pins 0 to 2, accelerometer values on three axes and an XYZ-combined magnetic field strength. The fact that two complex sensors are read via the I2C-bus is something that we can simply accept, as others have already done the difficult part of the work. Only if you insist on knowing exactly how this works do you

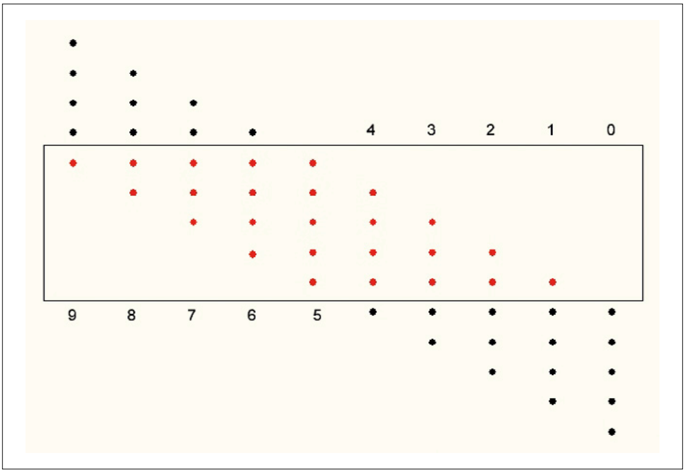


Figure 6. Numerical display.

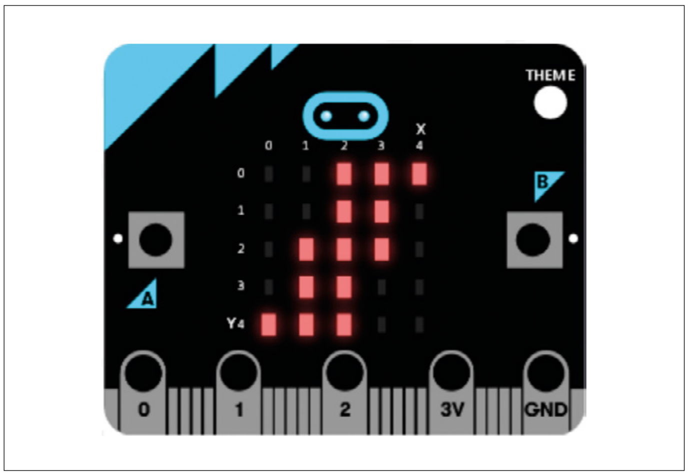


Figure 7. Displaying the figures 13579.

have to work through the deeper layers of countless embedded files.

All of the values measured are displayed in the terminal. While you are checking all of this out some ideas may occur to you of possible applications, such as measurements with magnets. Or else how about putting an accelerometer in your car to tell drivers when they are driving a little too friskily? The readout could look like this:

```
Time: 1715018 ms
Temp: 22 deg
P0 465 mV
P1 251 mV
P2 252 mV
X 0 mG
Y 1008 mG
Z 64 mG
B: 169311 uT
```

Static numerical display

An LED display with scrolling text is hard to read; it requires a lot of concentration. If you miss even one digit you'll have to wait until it comes around again. However, with 25 LEDs available we still have some other possibilities. Of course we are used to multi-digit static displays or alternatively to analog pointer instruments.

A static display can work with binary digits or in BCD code. In both cases you need four LEDs per digit. Since the Micro:bit has five LEDs in a column, however, there's another way of presenting digits. The figures 1 through 5 are displayed from bottom upwards using one to five LEDs. For the figures 6 through 9 the light-bar dangles downwards from the top. You can visualize this by imagining there are always five dots, of which some are not always visible. Our diagram makes this clear (**Figure 6**).

A display in this format (**Figure 7**) becomes readable intuitively after some practice. Incidentally this representation corresponds exactly with the structure of the Morse Code symbols for die figures 0 to 9, in which a lit LED indicates a dot and an unilluminated LED to a dash. So seven is sent as — — · · · in Morse.

For this kind of output format we have written a function in C. The figure to be output is transferred using the Integer Variables *n*. A voltage up to 3300 mV

Listing 3. Voltage measurement with static display.

```
//LED-Display
#include "MicroBit.h"
MicroBit uBit;

void ledDisplay (int n}
    int x;
    int y;
    int d;
    uBit.display.enable();
    MicroBitImage image(5,5);
    image.clear();
    for(int i = 0; i < 5; i++){
            d = 9 - n % 10;
            n = n / 10;
            x = 4 - i;
            for(int j = 0; j < 5; j++){
                  y = d - j;
                  image.setPixelValue(x,y,255);
            }
            }
            uBit.display.print(image);
        }


int main()
{
   uBit.init();
   uBit.io.P1.setDigitalValue(0);
   uBit.io.P2.setDigitalValue(1);

   MicroBitSerial serial(USBTX, USBRX);
   while (1) {
        int u = 3300 * uBit.io.P0.getAnalogValue()/ 1023;
        uBit.serial.printf("%d\r\n", u);
        ledDisplay (u);
        uBit.sleep(500);
        }
}
```

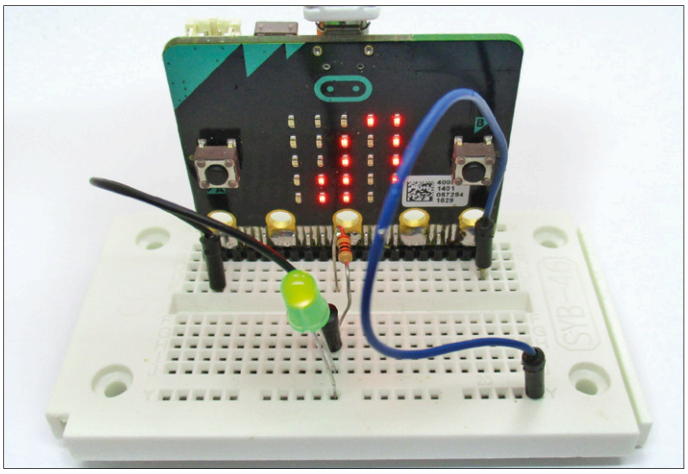


Figure 8. Measuring the LED voltage: 2.496 V.

can be indicated on the display (**Listing 3**).
In the main program two Port pins are switched additionally. P1 becomes Low, P2 is made High. You do not have to declare a changeover of data direction explicitly, as this happens automatically in the background. You can now investigate the load capability of the Port easily and use its own A-D converter for this straightaway. It's obvious that these have significantly higher impedance than the Ports of an AVR controller. Officially the Ports are capable of up to 5 mA in both directions. Test measurements indicate that a voltage drop of 300 mV arises. The 'on' resistance of the Port FETs thus amounts to around 60 Ω. Measurements taken with large loads indicate that the output current never rises above 15 mA. A current of around 10 mA flows when you connect an LED directly without a dropper resistor.
In normal situations you should use a dropper resistor, however, letting you then measure the LED voltage uniformly, compare differing LEDs and carry out further experiments (**Figure 8**).

(160273)

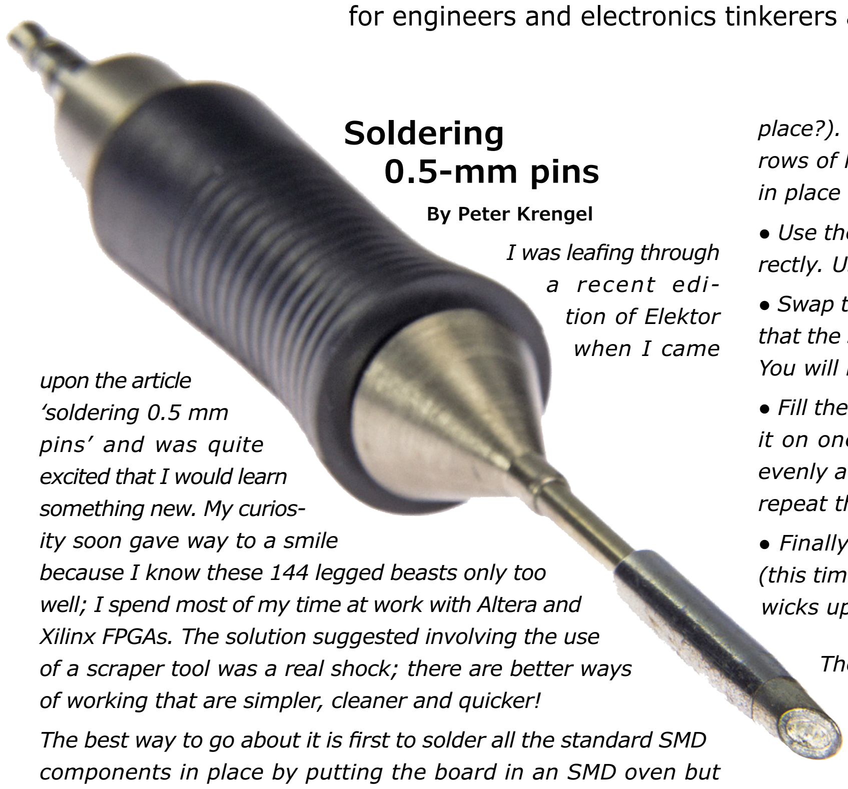
Web Links

[1] http://microbit.org/code/
[2] https://developer.mbed.org/
[3] https://lancaster-university.github.io/microbit-docs/ubit/
[4] https://developer.mbed.org/handbook/Windows-serial-configuration
[5] B. Kainka, BBC micro:bit Tests Tricks Secrets Code, CreateSpace 2016
[6] www.elektormagazine.com/150652
[7] www.elektormagazine.com/160274
[8] www.elektormagazine.com/160273

Tips and Tricks

From readers for readers

Here are some more neat solutions from our readers, sure to make life a little easier for engineers and electronics tinkerers alike



Soldering 0.5-mm pins

By Peter Krengel

I was leafing through a recent edition of Elektor when I came upon the article 'soldering 0.5 mm pins' and was quite excited that I would learn something new. My curiosity soon gave way to a smile because I know these 144 legged beasts only too well; I spend most of my time at work with Altera and Xilinx FPGAs. The solution suggested involving the use of a scraper tool was a real shock; there are better ways of working that are simpler, cleaner and quicker!

The best way to go about it is first to solder all the standard SMD components in place by putting the board in an SMD oven but leave out the beasts with thousands of feet. My simple approach works really well even when this first soldering stage is faulty and leaves the board with solder balls and splashes. The solution to the problem mounting the big chips is the 'Gullwing' soldering tip, available from Weller or Ersa [1],[2]. This type of bit has a dimple (or 'well') at its tip which acts as a tiny reservoir for the solder. I use this to 'drag solder' these large SMD chips with gull wing leads. You don't need to scratch the board during preparation because this process does not generate tiny balls of solder (usually caused by a poor or incorrectly adjusted SMD oven). Using this method you will be able to solder rows of 0.3-mm pins very cleanly without making any bridges. Here's how it goes:

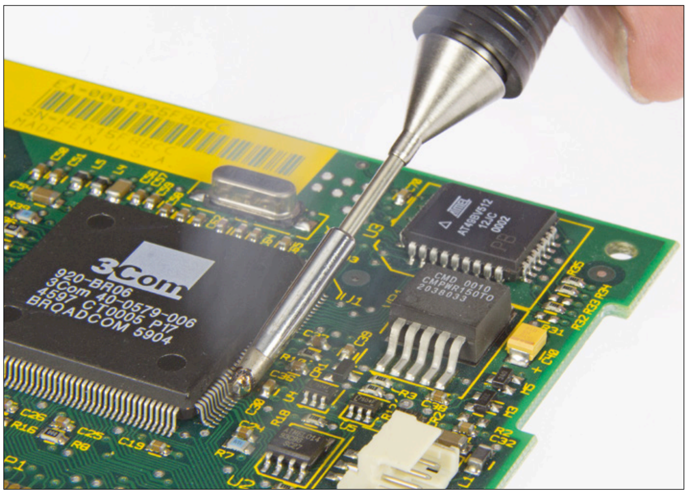
- *Rub the solder pads with 'No-Clean' flux and apply it also to the chip leads. I find this works best using a flux pen which first needs to be squeezed to bring the flux out onto the tip. Only gentle pressure is needed. Take care not to bend the leads as you apply the flux, you just need to dip each lead into the flux droplet at the end of the pen and then let it dry.*
- *Position the chip carefully on the board (pin 1 in the correct place?). Use an eye glass (10 × magnification) to position all the rows of leads in the centre of their pads. Solder the corner legs in place using a '0.2' soldering bit (e.g. Weller type RT).*
- *Use the eye glass again to check all the leads are lined up correctly. Use the soldering iron to reposition if necessary.*
- *Swap the 0.2 tip for the gullwing tip. Set the tip temperature so that the solder flows easily but not so hot that it starts to oxidize. You will need to experiment a bit to get the setting just right.*
- *Fill the hollow in the gullwing tip with solder then quickly place it on one corner of a row of pins and drag the bit slowly and evenly across the pins until one side is completed. Carry on and repeat this process for all four sides of the chip.*
- *Finally check for any solder bridges and use the gullwing tip (this time with an empty dimple) to drag over any bridges which wicks up excess solder.*

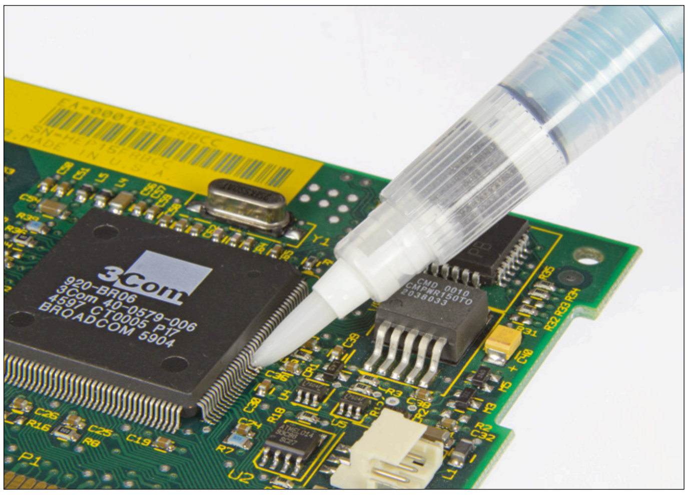
The best part of all: you will be able to completely mount (including inspection) a 144 pin chip in less than two minutes. There is no chance of overheating the joint because the solder bit only makes contact for a short period. Give it a go; you'll be surprised how easy it is!

The soldering looks so uniform and neat you would think it had been produced by a machine. In fact it is brighter than an oven baked version. This is probably because the joint is heated for such a short time the solder doesn't get a chance to start oxidizing, even if the bit temperature is set a little too high. In this state the solder has a high surface tension which ensures it covers the joint evenly. Liquids tend to form uniform shapes with the least surface area as witnessed in videos from the ISS showing water droplets in zero gravity. The final result is that the solder ends up where it should be; on and around the pads.

[1] Weller: www.weller.de/en/Weller--Products--Product-details.html?article_id=D0719205001379336391A111723

[2] Ersa Micro: www.kurtzersa.com/electronics-production-equipment/soldering-tools-accessories/soldering-desoldering-tips/soldering-tip-series-212/produkt-de





Messy IC reflow? Gullwing to the rescue

By Peter Krengel

The gullwing tip is also very handy at rescuing chips that have suffered a poor reflow procedure. For this you just need a No-Clean flux pen or dropper.

- *Rub the No-Clean flux pen around all the leads of the soldered IC so that some of the excess wicks underneath the chip.*
- *Immediately after applying the flux (it dries really quickly) drag the gullwing solder bit along each row of leads in turn, in this case ensure that the dimple in the solder bit is empty before you start.*
- *You're done!*

Instead of the No-Clean flux pen you can also use just standard rosin, it works out much cheaper. For this you need a wooden match (used), a good clean toothbrush and a bottle (100 ml) of 99 % pure ethanol (not for drinking, just for cleaning up!). This small quantity should be available from a drug store or chemist but it's probably cheaper via an online auction site. Don't use methylated spirit or isopropanol, these leave white spots on the board which can be tricky to remove.

Start by warming up some rosin in a can until its liquid. Dip the matchstick into the flux and transfer a drop of flux onto the leads and distribute it around all the leads. You need to work quickly here otherwise the rosin will start to set, it's a good idea to preheat the board using a hair dryer or better still a hot air gun set to 120 to 130 °C.

As before, drag the empty gullwing solder tip along each row of pins slowly and evenly, if necessary repeat this step. All of the excess solder and bridges should now be gone leaving shiny joints on all the pads.

The toothbrush and alcohol now come into play. Unlike No-Clean flux when you work with rosin you need to clean up afterwards with ethanol. After drag soldering the IC pins are usually swimming in flux, before you start to clean up the board needs to cool down. You can pop it in a freezer for five minutes if necessary.

Excess rosin can then lightly crushed using the match (prevents the PCB from getting scratched). Remove any loose rosin debris by tapping the board or with an air gun then finally clean up with plenty of alcohol and a toothbrush — job done!

Flux by any other name

By Peter Krengel

This is just a tip for those who think they are paying too much for too little rosin. Electronics is not the only industry that makes use of it. It's a resin extracted from pine trees, the same stuff you rub on a violin bow to stop it skidding over the strings. The butchery trade also uses it for pulling out feathers from poultry and bristles from pig's skin. Pure rosin can be purchased through online auction sites where it's available as blocks, flakes or a powder. Make it pourable by warming it up and store it in small jars that have been cleaned out beforehand with detergent and alcohol.

Wick revival

By Peter Krengel

Rosin is also a useful reviver for tarnished solder wick. First off soak the wick in a dilute acid solution (5–10 % hydrochloric acid which the building trade use for cleaning masonry) to remove any copper oxide. Give the resulting shiny wick a thorough rinse in running water and draw it through some liquefied rosin. As it cools the rosin sets and you can break off any excess. You'll be amazed at how much better it works once it's been revived in this way. ◄

(160324)

Have you come up with an inspired way of solving a really challenging problem? Or found an ingenious but 'alternative' way of using some component or tool? Maybe you've invented a better or simpler way of tackling a task? Do write in – for every tip that we publish, we'll reward you with UKP40 (or local equivalent)!

eLABS PROJECT

Microphone Preamp

with professional specs

By **Joseph Kreutz** (Germany)

LY FLEX J2 4UU.5 ROHS CE 03/16

Tape recorders have been obsolete for ages now, replaced with laptops fitted with a good-quality 'USB sound card'. Now while a PC's 'line' input generally offers acceptable characteristics, 'mic' inputs typically have only mediocre noise and distortion levels. Above all, they offer only a mono signal and are not suitable for connecting professional microphones with a balanced output, which often also require 48-V "phantom" powering. Here I'm suggesting a preamplifier that gets around these two shortcomings and is bound to please those who own 'top range' sound cards.

Amplifier circuit

Figure 1 gives the preamp circuit. Although it might seem outdated, the use of a transformer (TR1) makes it possible to design a simple circuit with effective input isolation and good common-mode signal rejection. The transformer substantially reduces interference, such as that caused by cellphones. It's easy to find microphone transformers with a 1:10 ratio. This type of transformer steps the signal voltage up by 20 dB, which considerably improves the preamp's signal-to-noise ratio.

In order to preserve all these benefits, we need a FET-input opamp with negligible noise current. The AD743 opamp is ideal because its input noise is very low (2.9 nV/√Hz) [1], but it is expensive. The cheaper OPA134 from Burr-Brown (TI) is an excellent choice; however, for the preamp prototype, I used a TL071 with astonishingly good results [2]. Dual opamps like the OP2134 or TL072 will be perfect for stereo projects.

To buy the transformer, take a look at Bürklin [3], who offer models with good value for money. Lundahl [4], Sow-

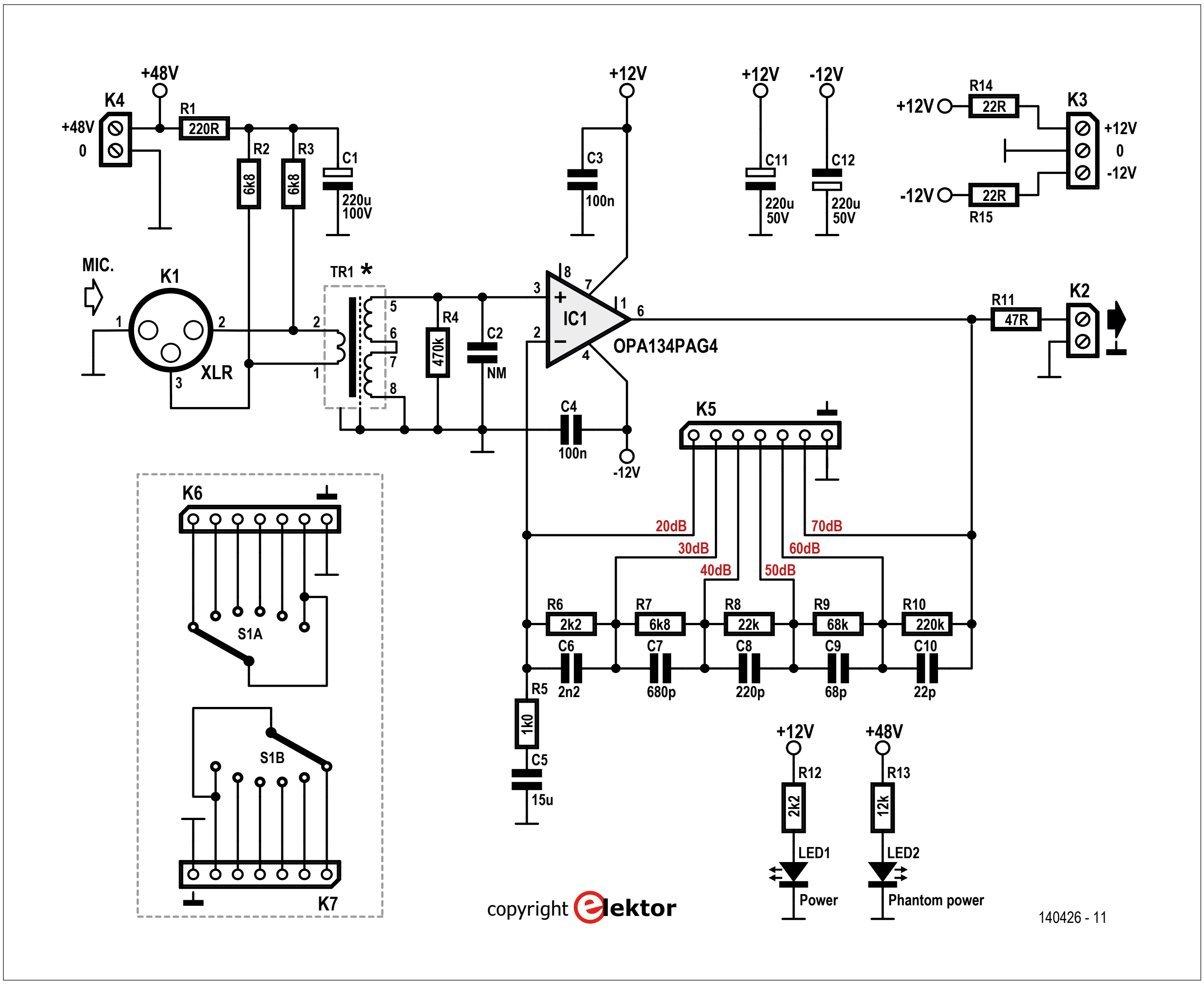


Figure 1. The heart of this amplifier is the opamp.

ter [5], and Jensen [6] offer professional-quality products, but are dearer. And don't be afraid to look elsewhere on the 'net. Note that the PCB can accommodate several models for TR1: ÜP3095M and ÜP3096M from Pikatron, LL1935 from Lundahl, and JT-115K-EPC from Jensen. The Lundahl LL1935 transformer has two primary windings and two secondary windings [note from e-Labs: it may happen that one of the two primary windings gets connected in the reverse direction causing a short circuit, so read that datasheet].

Let's get back to the circuit: to apply the signal to TR1 primary, the mic should preferably be connected using a connector to the usual professional XLR standard or via a 6.35-mm / 0.25" stereo jack. In this case, the tip of the jack corresponds to the "hot" (left channel), the ring to the 'cold' (right channel), and the remainder to Ground. Condenser mics require a 48-V supply voltage, applied via 6.8-kΩ resistors R2 and R3. The precise value of R2 and R3 is not essential, but these two resistors must be matched to 0.4% or better [7, 8]. Get out your hi-res ohmmeter to pair them. They must be rated at 0.5 W at least to avoid their burning out in the event of a short to ground.

One side of TR1 secondary is connected to ground, and the other to the + input of the opamp (IC1); the — input is connected to the feedback network that sets the preamp gain. The 6-way rotary switch connects the opamp output to its — input, either directly or via resistors R6–R10 in series. Each step increases the gain by 10 dB. As the transformer already raises the level by 20 dB, the preamp gain can be varied between 20 and 70 dB. It's crucial for the switch to be a 'make-before-break' type: the new connection is made before the previous one is broken. This makes it possible to avoid abrupt jumps in the gain, or even temporarily maximum gain, with resulting howlround. The capacitors in parallel with R6–R10 limit the preamp bandwidth to 32 kHz. These are high-quality polystyrene axial types. However, the PCB layout also lets you fit ceramic or plastic film capacitors. On TR1 secondary, you have space if necessary to fit a capacitor (C2) intended to damp unwanted transformer resonance. C2 can usually be omitted.

A few more tips:
Decoupling capacitors C3 and C4 (100 nF) must be fitted as close as possible to the opamp. I would recommend using 1% metal film resistors; this is essential for R5–R10. R4 (470 kΩ) can be omitted if the mic does not require a specific load impedance. Otherwise, you need to choose a value one hundred times the recommended impedance. The electrolytic capacitors must be rated at no less than 25 V, except for C1, which must be a 63 V or 80 V type. The opamp supply voltages are not critical, so long as they meet the manufacturer's specifications.

If you use only dynamic microphones, i.e. without a 48-V supply, components C1 and R1–R3 are not required.

Capacitor C5 has a value of 15 µF, but for a lower cut-off frequency (or less phase shift in the low frequency range), a 22-µF capacitor from the same range is suitable.

±12 V supply

The opamp (IC1) is powered using the +12 and −12 V rails produced by the lower part of the circuit in **Figure 3**. This is a perfectly conventional power supply built using two regulator ICs. Note that it's wise to fit the 100-nF decoupling capacitors as close as possible to the regulators. A bridge of 1N4007 diodes provides the rectification. Fuse F5 that protects transformer TR1 is rated according to the manufacturer's recommendations. LED1 fitted on the PSU front panel shows when the unit is operating. Certain examples of the 7912 regulator only give their nominal voltage when they are loaded by a minimum current of 5 mA. So if LED1 is not fitted, it's a good idea

PROJECT INFORMATION

audio
preamplifier
microphone

➡ entry level
intermediate level
expert level

5 hours approx.
(excluding fitting into case)

nothing special

€270 / £235 / $290 approx.
(including cases)

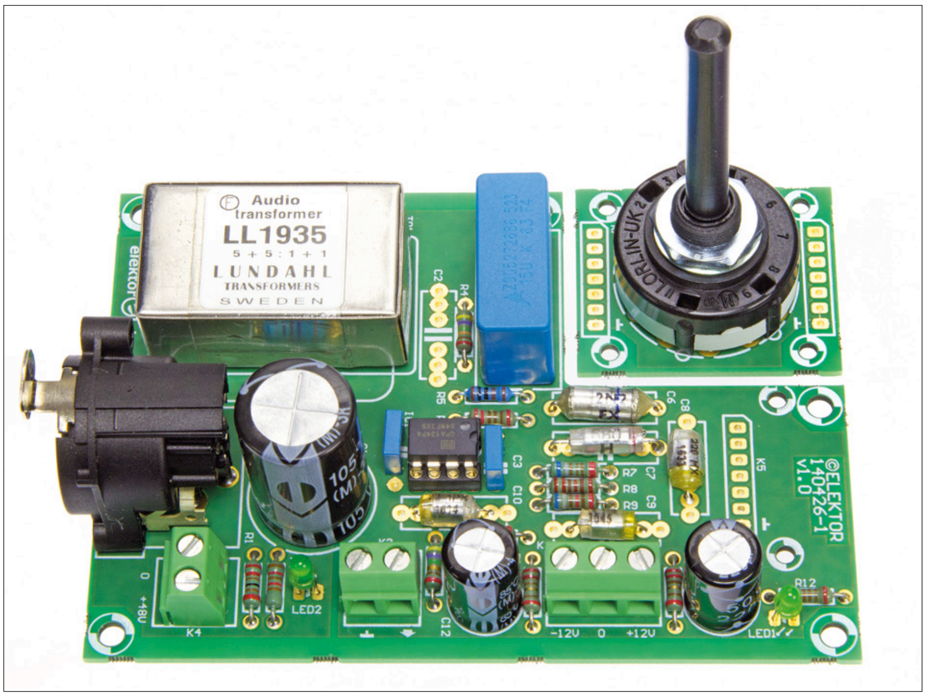


Figure 2. Populated amplifier PCBs.

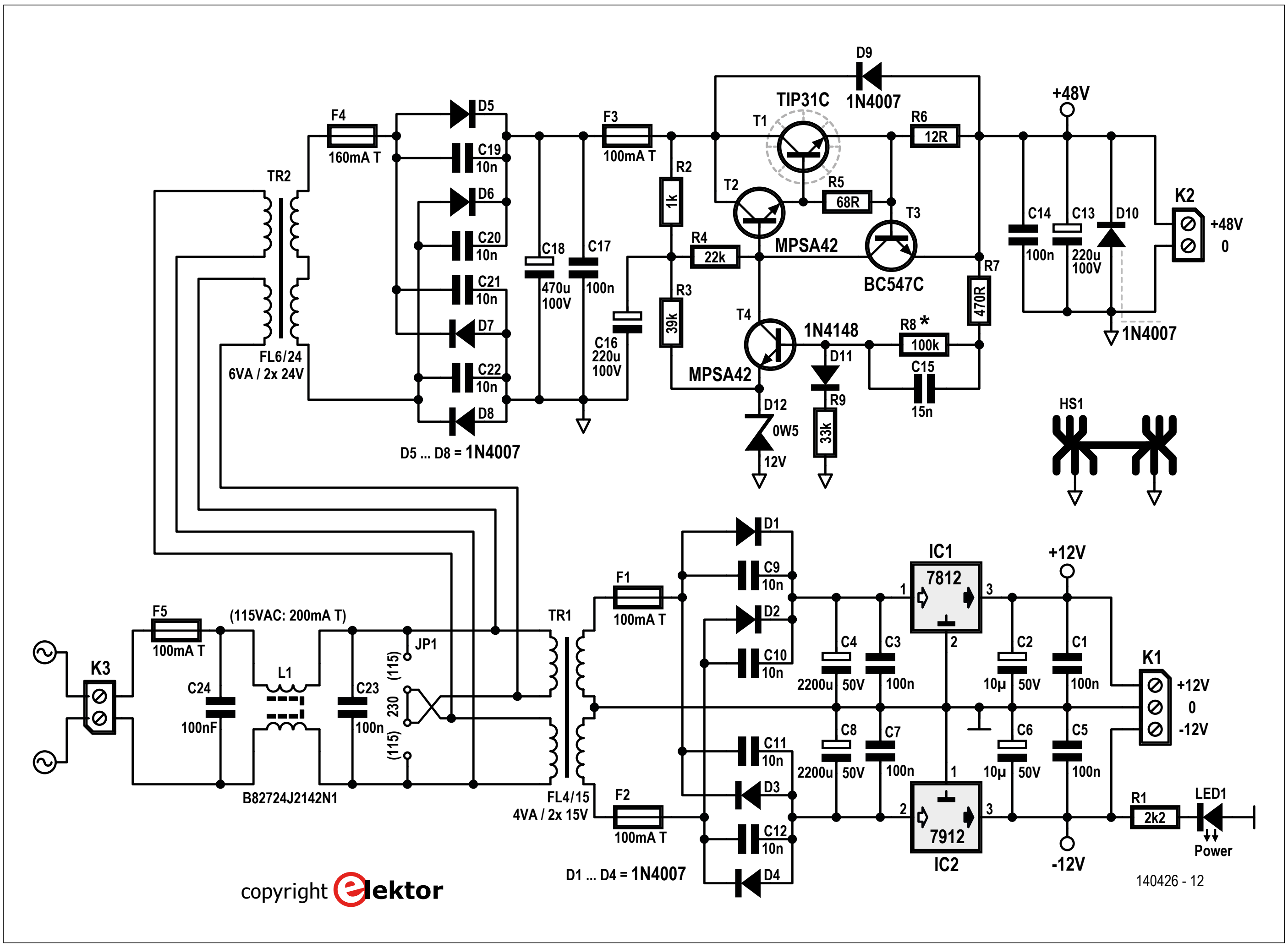


Figure 3. Circuit of the dual power supply: ±12 V and 48 V.

to fit a 2.2-kΩ resistor from the output of regulator IC2 to ground. The OPA134 opamp draws a maximum of 5 mA @ T_{amb} = 25°C. Hence the regulators don't need heatsinks, even when two amplifiers are connected.

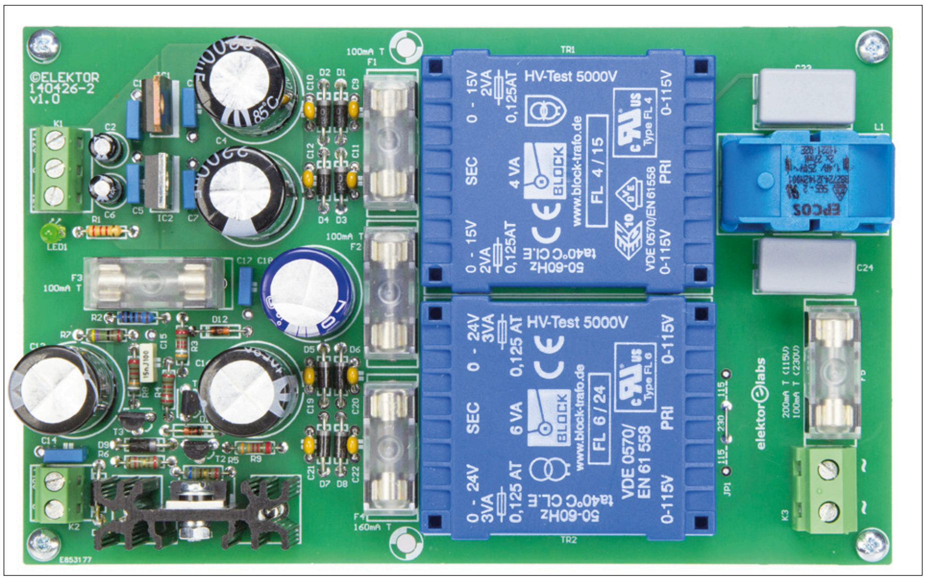


Figure 4. Populated dual power supply PCB.

Desktop computers all produce +12 and −12 V voltage rails that can be used to power the preamp.

If you go for this solution, filter these voltage rails carefully: they will certainly be polluted by the noise produced by the motherboard and other equipment. It's wise to insert a 100 mA time-delay fuse in each power rail so as not to damage the computer in the event of a problem.

48 V supply

The top part of Figure 3 shows the power supply that provides the 48 V rail if needed. The EN 61938 standard [8] make provision for a maximum mic current of 10 mA, and tolerates a voltage variation of ±4 V around the nominal value. The AC line voltage is applied to transformer TR2 which supplies 2 × 24 V.

The regulator is based on a conventional circuit and uses only discrete components. The 12-V zener diode establishes

COMPONENT LIST – PREAMPLIFIER

Resistors

All metal film
R1 = 220Ω, 1%, 250 mW
R2, R3 = 6.8kΩ, 0.1%, 500mW
R4 = 470kΩ, 1%, 400mW
R5 = 1kΩ, 1%, 600mW
R6,R12 = 2.2kΩ, 1%, 400mW
R7 = 6.8 kΩ, 1%, 400mW
R8 = 22kΩ, 1%, 250mW
R9 = 68kΩ, 1%, 250mW
R10 = 220kΩ, 1%, 250mW
R11 = 47Ω, 1%, 250mW
R13 = 12kΩ, 1%, 250mW
R14, R15 = 22Ω, 1%, 250mW

Capacitors

C1 = 220µF, 100V, 20%, pitch 5/7.5mm, $\varnothing_{max}$ 17.8mm
C2 = not fitted
C3,C4 = 100nF, 100V, 5%, pitch 5/7.5mm, PET
C5 = 15µF, 63 V, 10%, pitch 5/7.5/10/15/22.5mm, PET
C6 = 2.2nF, 1%, 160V, pitch 5/7.5/10/14.6mm, polystyrene, axial
C7 = 680pF, 2.5%, 160V, pitch 5/7.5/10/14.6mm, polystyrene, axial
C8 = 220 pF 2.5%, 160V, pitch 5/7.5/10/14.6mm, polystyrene, axial
C9 = 68pF 2.5%, 160V, pitch 5/7.5/10/14.6mm, polystyrene, axial
C10 = 22pF ±1pF, 160V, pitch 5/7,5/10/14,6mm, polystyrene, axial
C11, C12 = 220µF, 50V, 20%, pitch 5mm, $\varnothing_{max}$ 10mm

Transformer

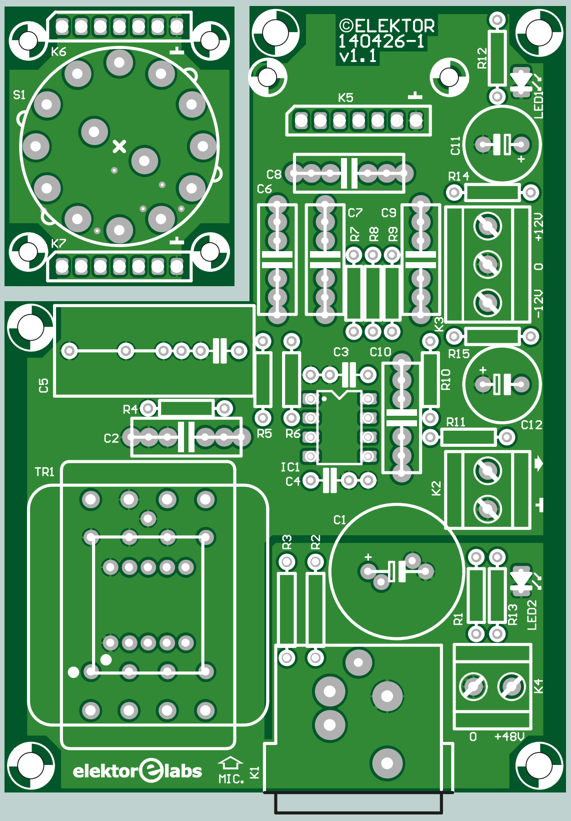
TR1 = LL1935, 1:10 (1:1:5+5), Lundahl, 38 × 23 × 16 mm

Semiconductors

LED1 = green, 3mm, T-1
LED2 = blue, 3mm, high intensity, T-1
IC1 = OPA134PAG4, DIP-8

Miscellaneous

K1 = audio XLR connector, 3 contacts, female, PCB mount (Neutrik NC3FAAH1)
K2, K4 = 2-way PCB screw terminal block, 0.2" pitch, 630V
K3 = 3-way PCB screw terminal block, 0.2" pitch, 630V
K5 = 7-pin pinheader, angled, 0.1" pitch



K6, K7 = not fitted (pos. opposite K5)
S1 = 2-pole, 6-way rotary switch, make-before-break, Lorlin type CK1060
PCB # 140426-1
Case type 1455K1201, Hammond Manufacturing

Other transformers that fit the board:

TR1 = Pikatron ÜP 3095 M, 1:10, 18×14,5×13,2 mm (see [3])
TR1 = Pikatron ÜP 3096 M, 1:15, 18×14,5×13,2 mm (see [3])
TR1 = Jensen transformers JT-115K-EPC, 1:10, 31.75×31.75×23.62 mm (see [6])

the reference voltage; to regulate the voltage, the feedback signal is applied to transistor T4 via R7 and R8. Transistor T1, a TIP31C, is generously rated to withstand mistreatment. Moreover, T1 is fitted on a heatsink, as it heats up because of the high regulator input voltage. A thermal resistance of around 20°C/W will be enough to keep T1 cool. As a last resort, the 100-mA time-delay fuse F3 will certainly limit the damage in desperate situations…

Transistor T3 and resistor R6 limit the current produced by this power supply to 50 mA, which is amply sufficient for several mics. The electrolytic capacitors in the 48-V supply must have a working voltage of 100 V. And lastly, the Elektor lab has measured residual ripple below 0.2 mV_{pp} at 10 mA, which is eliminated by the amplifier board.

Putting it all together

Transformer TR1 and TR2 primaries can handle both nominal AC line voltages: 115 V and 230 V. For 230 V, a wire strap is needed for JP1; for 115 V, two are needed. Attention: all three straps are never used together, this is clearly indicated on the PCB. For 115 V, the fuse rating must be doubled. The transformer manufacturer only recommends fitting a fuse on the secondary. However, for greater safety, we've added one on the primary too.

To house the two PCBs, the Elektor lab has selected two cases from the 1455 series by Hammond (see photo at top of article). The preamp is in the smaller case. Attention: the metal case of the Jensen transformer may short across the PCB tracks: when you are soldering this component, leave a gap between it and the PCB or fit a 0.5-mm thick insulating sheet.

Safety: the cable from the power outlet is connected directly to the PCB and there must be no contact with the case, to ensure insulation to Class II. All wires, pins, and PCB tracks that are directly or indirectly connected to AC line voltage must be at least 6 mm from the metal parts of the case. Underneath the PCB, cut off the wires and pins carrying mains voltage as short as possible. For the supply wiring, you must use 0.75 mm² flexible wire.

All that remains is to make up the cable to connect the preamp to the PSU. This is fitted with a 5-pin XLR connector; this may seem a bit over the top. Of course, there are plenty of other types of connector you can use too. Another solution is to use two separate cables, one for the symmetrical ±12 V rails and the other for the phantom powering. ⏮

(140426)

COMPONENT LIST – POWER SUPPLY

Resistors

R1 = 2.2kΩ, 5%, 0.25W, carbon
R2 = 1.0kΩ, 1%, 0.6W, metal film
R3 = 39kΩ, 1%, 0.25W, metal film
R4 = 22kΩ, 1%, 0.25W, metal film
R5 = 68Ω, 1%, 0.4W, metal film
R6 = 12Ω, 1%, 0.25W, metal film
R7 = 470Ω, 1%, 0.25W, metal film
R8 = 100kΩ, 1%, 0.25W
R9 = 33kΩ, 1%, 0.25W

Capacitors

C1,C3,C5,C7,C14,C17 = 100nF, 100V, 5%, pitch 5/7.5mm, PET
C2,C6 = 10µF, 50V, 20%, pitch 2mm, Ømax 6.5mm.
C4,C8 = 2200µF, 50V, 20%, pitch 5/7.5mm, 17.8mm max.
C9,C10,C11,C12,C19,C20,C21,C22 = 10nF, 200V, 10%, pitch 0.2", X7R ceramic
C13,C16 = 220µF, 100V, 20%, pitch 5/7.5mm, 17.8 mm max.
C15 = 15nF, 100V, 5%, pitch 5/7.5mm
C18 = 470µF, 100V, 20%, pitch 5/7.5mm, 17.8mm max.
C23, C24 = 100nF, X1, 440VAC, pitch 10/125/15 mm, polypropylene

Inductors & Transformers

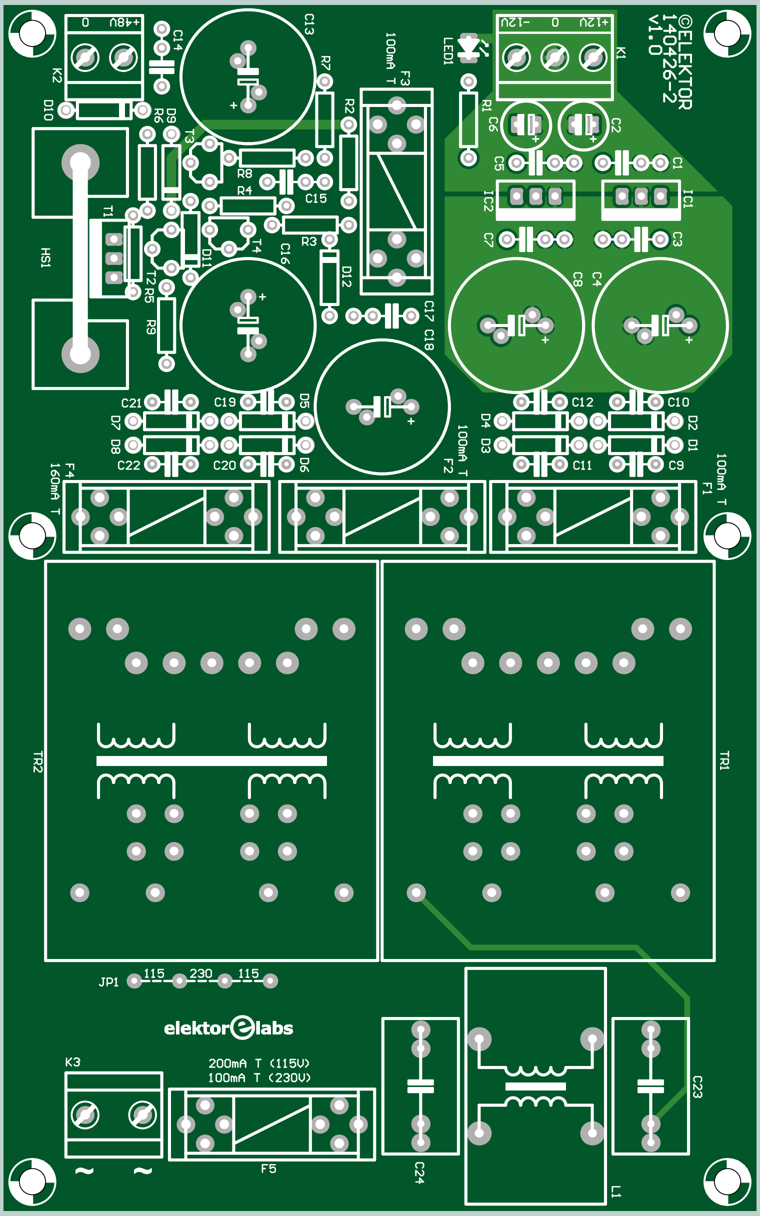
L1 = B82724J2142N1, choke, common mode, 2×27mH, 1.4A
TR1 = Block type FL4/15, prim. 2×115V, sec. 2×15V, 2VA, PCB mount
TR2 = Block type FL6/24, prim. 2×115V, sec. 2×24V, 6VA, PCB mount

Semiconductors

D1,D2,D3,D4,D5,D6,D7,D8,D9,D10 = 1N4007, 1000V, 1A, DO-41
D11 = 1N4148, 100V, 200mA, DO-204AH
D12 = BZX55C12, zener diode 12V, 0.5W, DO-35
LED1 = green, 3mm, T-1
T1 = TIP31C, NPN, TO220 (100V / 40W / 3A)
T2, T4 = MPSA42, NPN, TO-92 (300V / 625mW / 500mA)
T3 = BC547C, NPN, TO-92 (45V / 500mW / 100mA)
IC1 = 7812, TO-220-3
IC2 = 7912, TO-220AB-3

Miscellaneous

JP1 = 0.6mm diam. wire (see text)
K1 = 3-way PCB screw terminal block, 0.2" pitch, 630V
K2 = 2-way PCB screw terminal block, 0.2" pitch, 630V
K3 = 2-way PCB screw terminal block, 0.3" pitch, 630V
F1, F2, F3, F5 (*1) = fuse, 100mA, slow blow, 250V, 20×5mm
F4 = fuse, 160mA, slow blow, 250V, 20×5mm
F1,F2,F3,F4,F5 = fuse holder, 20×5 mm, 500V, 10A
F1,F2,F3,F4,F5 = fuseholder cap , 20×5 mm



HS1 = heatsink, Aavid Thermalloy type SW25-4, 13°C/W, 34.5×12.5×25 mm
PCB # 140426-2
*1. For 115 VAC : 200mA, slow blow, 250V, 5×20 mm
Case type 1455N1601, Hammond Manufacturing

FROM THE STORE
➞140426-1
PCB
➞140426-2
PCB

XLR connectors and cable for connecting the power supply to the preamplifier		
PSU	**Mic preamp**	**Cable**
Female XLR Neutrik NC5FD-LX, chassis-mounting (Farnell 1390124)	Male XLR Neutrik NC5MP, chassis-mounting (Farnell 250820)	Elektor lab choice: Pro Power PPCY4C0.5 (by the meter), screened, 4 cores 0.5 mm², $\varnothing_{ext}$ 6.5 mm
Male XLR Neutrik NC5MX, cable-mounting (Farnell 250764)	Female XLR Neutrik NC5FXX, cable-mounting (Farnell 1108204)	Alternative: Pro Power PPCY4C0.5 (by the meter), screened, 4 cores 0.75 mm², $\varnothing_{ext}$ 7 mm

Curves measured using Audio Precision Analyzer @ Elektor Labs

Conditions in both cases:

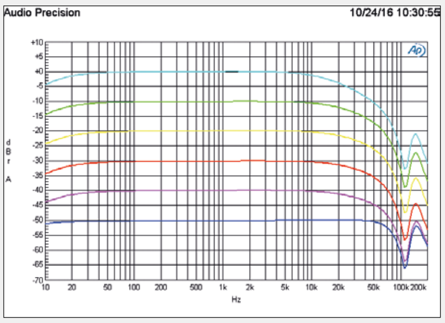
- source impedance = 200 Ω;
- min. gain = 19.4 dB;
- max. output voltage = 7.8 V (THD 0.1%, gain 70 dB)

Curve color according to total gain:
70 dB (light blue), 60 dB (green), 50 dB (yellow), 40 dB (red), 30 dB (pink), and 20 dB (dark blue).

Table 1. Measurements using **Lundahl LL1935**			
Gain	**THD+N (source 200 Ω) 1 kHz, 1 V_{out}, B 22 kHz**	**S/N (source = 200 Ω) relative to 1 V_{out}, B 22 kHz**	**Lower/upper cutoff frequency @ −3 dB**
20	0.0032 %	110 dB / 112.7 dBA	< 10 Hz / 74 kHz
30	0.0014 %	100 dB / 103 dBA	12.2 Hz / 37 kHz
40	0.0028 %	90.8 dB / 93.1 dBA	12.8 Hz / 34 kHz
50	0.0087 %	80.8 dB / 83 dBA	12.8 Hz / 34 kHz
60	0.027 %	71 dB / 73 dBA	12.8 Hz / 30.6 kHz
70	0.078 %	62 dB / 63.5 dBA	12.8 Hz / 17 kHz

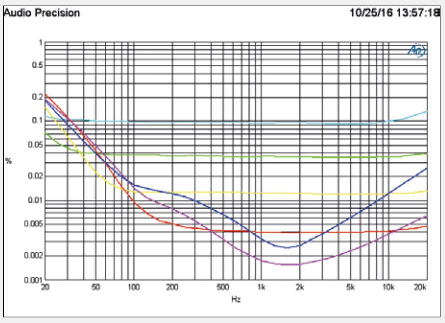
Amplitude as a function of frequency for the various gain values, over the range 10 Hz –200 kHz.

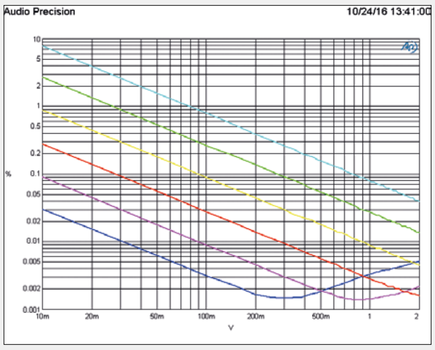
At 20 dB (dark blue), the curve is almost the same as for the transformer itself. IC1 acts simply as a buffer. At 151 kHz there is a resonance peak, the amplitude and frequency of which are highly dependent on the source impedance and type (unbalanced or balanced). For the gains of 30, 40, and 50, the cutoff frequencies are more or less the same, but at 60 and 70 dB, the opamp's unity gain bandwidth begins to be make itself felt. With the gain at 50 dB, the OPA134's bandwidth is equal to 8 MHz / 316, i.e. 25.3 kHz! If you want greater bandwidth with the highest gains, the LME49710 (gain/bandwidth product = 55 MHz) or OPA627 (gain/bandwidth product = 16 MHz) are possible options. Replacing capacitor C5 by a 22-µF type will lower the cut-off frequency to 9.6 Hz and the gain with respect to 20 Hz from −1.48 dB to −0.87 dB.



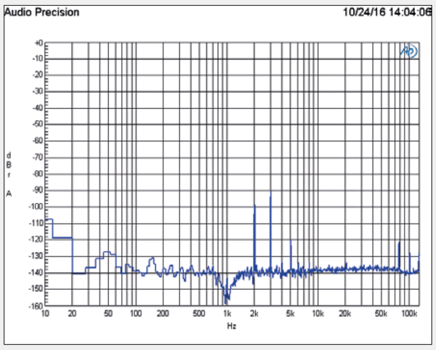
Distortion + noise as a function of frequency for V_{out} = 1 V and the various gain values over a bandwidth of 80 kHz.

To maintain the output voltage constant irrespective of the gain value, the input signal level is increased accordingly. For the highest gain, an input of 335 µV is sufficient obtain 1 V at the output. With the highest gain, there is above all noise, but just as for the lower gains, the input voltage increases the distortion and becomes the main factor. The increase in distortion at the end of the curves for the highest input levels is solely due to the transformer.

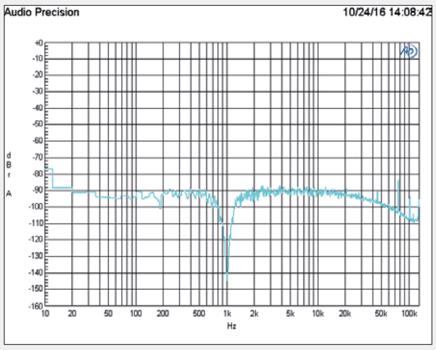




Distortion + noise as a function of output level @ 1 kHz and for a bandwidth of 22 kHz.
The distortion at the ends of the curves rises for the highest gains.



Fast Fourier Transform (FFT) @ 1 kHz for the lowest total gain (20 dB) and V_{out} = 1 V.
It is primarily a 3rd harmonic @ −90 dB that is responsible for the ‘total harmonic distortion + noise’ result of 0.0032%.

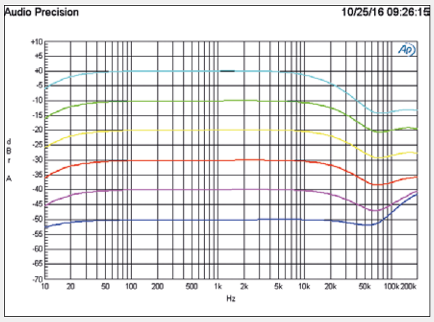


FFT @ 1 kHz for a higher total gain (70 dB) and V_{out} = 1 V.
As all the frequencies (for that, read “noise”) approach the 90 dB-down level, ‘total harmonic distortion + noise’ is equal to 0.078%.

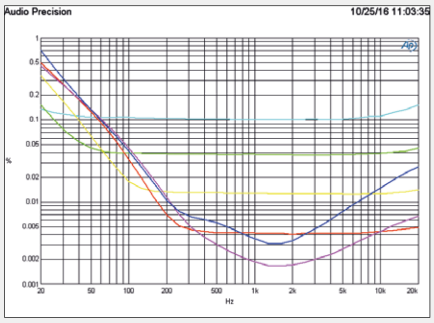
Table 2. Measurements using **Pikatron ÜP3095M**

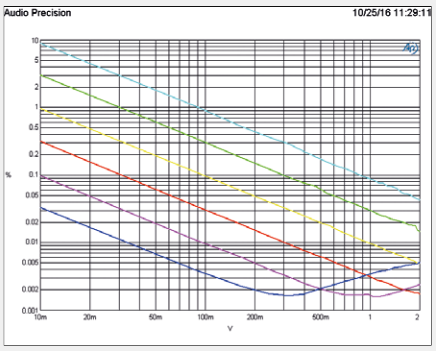
Gain	**THD+N (source 200 Ω) 1 kHz, 1 V_{out}, B 22 kHz**	**S/N (source = 200 Ω) relative to 1 V_{out}, B 22 kHz**	**Lower/upper cutoff frequency @ −3 dB**
20	0.0034%	109 dB / 112 dBA	< 10 Hz / ? >200 kHz)
30	0.0016%	100 dB / 102 dBA	14.5 Hz / 30 kHz
40	0.0031%	90.6 dB / 92.6 dBA	15.3 Hz / 28.3 kHz
50	0.0097%	80 dB / 82.2 dBA	15.3 Hz / 26.2 kHz
60	0.030%	70.4 dB / 72.5 dBA	15.3 Hz / 23.3 kHz
70	0.086%	61.1 dB / 62.7dBA	15.3 Hz / 16.2 kHz

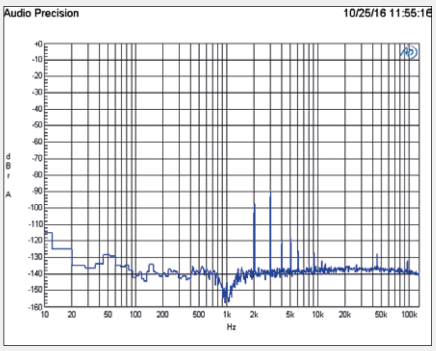
Amplitude as a function of frequency for the various gain values, over the range 10 Hz –200 kHz.
There is a resonance peak just beyond 200 kHz. The total bandwidth is lower than that observed with the Lundahl transformer, but satisfactory for practical purposes. Replacing capacitor C5 by a 22 μF one will lower the cut-off frequency from 15.3 to 12 Hz.

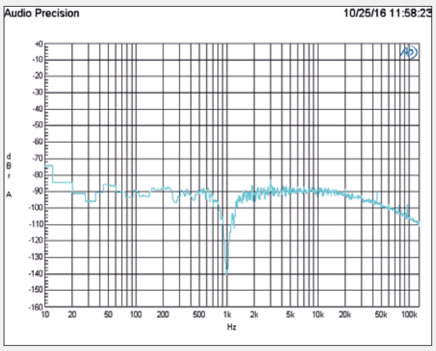


Distortion + noise as a function of frequency for V_{out} = 1 V and the various gain values over a bandwidth of 80 kHz.
Only at the lowest frequencies is the sum of “total distortion + noise” higher than that seen with the Lundahl transformer.




Distortion + noise as a function of output level @ 1 kHz and for a bandwidth of 22 kHz.
It is difficult to see any difference from the curve for the Lundahl. The levels are slightly higher.


Fast Fourier Transform (FFT) @ 1 kHz for the lowest total gain (20 dB) and V_{out} = 1 V.
The 3rd harmonic is the key factor and the rest of the harmonics are a little higher than those seen on the curve for the Lundahl. The sum 'total harmonic distortion + noise' here is 0.0034%.


FFT @ 1 kHz for a higher total gain (70 dB) and V_{out} = 1 V.
The 3rd harmonic is barely visible and the rest is only noise. The sum 'total harmonic distortion + noise' here is 0.086%.

Analysis of the preamplifier noise

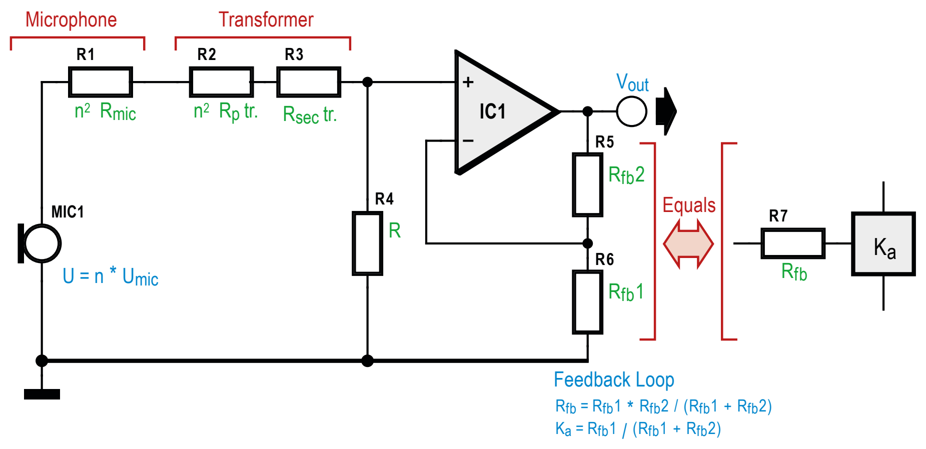


Figure. Equivalent circuit used for analyzing the preamp noise.

1. Theory
The noise sources taken into consideration are, on the one hand, the inherent thermal noise of the resistive components, and on the other, the noise produced by the active elements, here the operational amplifier. No account is taken of other sources of noise, such as those resulting from electromagnetic or electrostatic fields. Besides, it is usual to reduce the degree of nuisance from these, e.g. by using appropriate shielding.
The **figure** represents the equivalent circuit used for analyzing the preamplifier noise, arising from two sources: the thermal effects in the various resistors, together with the component caused by the active component, here the opamp. Only the real parts of the impedances effectively contribute to the noise.

The signal source comprises the mic, which presents the internal resistance R_{mic} and produces the signal U_{mic}, applied via transformer T1 to the + input of opamp U1. Each winding of the transformer presents an internal resistance: $R_{p\ tr}$ for the primary, $R_{sec\ tr}$ for the secondary. The transformer is also characterized by the transformation ratio n between the primary and the secondary. Seen from the opamp's + input, the total resistance of the branch of the circuit comprising the mic and transformer is thus equal to:

$$R_a = n^2 \cdot \left(R_{mic} + R_{p\,tr}\right) + R_{sec\,tr} \qquad (1)$$

Where applicable, this resistance is in parallel with R4, where used. The resulting total resistance is:

$$R_{inp\ pos} = \frac{1}{\frac{1}{R_a}+\frac{1}{R_4}} = \frac{1}{\frac{1}{n_2\cdot\left(R_{mic}+R_{p\ tr}\right)+R_{sec\ tr}}+\frac{1}{R_4}} \quad (2)$$

The opamp − input is connected to the resistances $R_{fb\ 1}$ and $R_{fb\ 2}$. This feedback network is characterized by an attenuation K_a and an equivalent resistance of, respectively:

$$K_a = \frac{R_{fb_1}}{R_{fb_1}+R_{fb_2}} \quad (3)$$

and

$$R_{fb} = \frac{1}{\frac{1}{R_{fb1}}+\frac{1}{R_{fb2}}} \quad (4)$$

Hence at the opamp inputs, the thermal noise will be:

$$e_{n\ th} = \sqrt{4\cdot k_b\cdot T\cdot\left(R_{inp\ pos}+R_{fb}\right)\cdot B} \quad (5)$$

where k_b is the Boltzmann constant, T the temperature in kelvins, and B the bandwidth of the signal.
The noise voltage attributable to the opamp will be:

$$e_{n\ comp\ actifs} = \sqrt{\left\{e^2_{n\ op}+\left[i_{n\ op}\cdot\left(R_{inp\ pos}+R_{fb}\right)\right]^2\right\}\cdot B} \quad (6)$$

where $e_{n\ op}$ and $I_{n\ op}$ are the noise voltage and current densities respectively published in the datasheets for this device. The final result is an overall noise voltage:

$$e_{n\ totale} = \sqrt{e^2_{n\ th}+e^2_{n\ comp\ actifs}} \quad (7)$$

As the noise sources are not correlated, the voltage resulting from combining them is not calculated by simple addition, but by taking the square root of the sum of their amplitudes squared, which amounts to saying that in this case, it is the individual powers that come into play.

2. Example with figures

Let's look at the component values of the circuit in Figure 1 in order to compare four opamps: OPA134 from Burr Brown, AD743 from Analog Devices, NE5534, and TL071. We are considering a bandwidth of 30 kHz — roughly that of the preamplifier.
Since the author of the article has a clear preference for hot summer afternoons, the temperature will be 300 K, or 26.85 °C if you prefer. The transformation ratio n is 10; the primary and secondary resistances are 50 and 1,500 Ω respectively. We are assuming that the mic presents an internal resistance of 100 Ω — a realistic value for this type of device. The noise level is calculated using the highest value of R_{fb}, corresponding to maximum gain.
On the preamp prototype, R_a is not used. The thermal noise due to the resistances is calculated using equations (1), (4), and (5):

$$R_{inp\ pos} = 10^2\cdot\left(100+50\right)+1500 = 16500$$

and

$$R_{fb} = \frac{1}{\frac{1}{1000}+\frac{1}{319\cdot10^3}} = 996.87 \approx 997$$

These values are entered into eq. (5):

$$e_{n\ th} = \sqrt{4\cdot1.38\cdot10^{-23}\cdot300\cdot\left(16500+997\right)\cdot30000}$$

After calculation, we obtain:

$$e_{n\ th} = 2.95\cdot10^{-6}\ \text{V}$$

The opamp's contribution to the noise is determined using eq. (6), and its total value referred back to the input using eq. (7). The results for the four opamps are shown in a table, to make it easier to compare. The signal-to-noise ratio in the bottom line of the table is given for a voltage of 5 mV out of the microphone.

opamp	**AD743**	**OPA134**	**NE5534**	**TL 071**
$e_{n\ op}$ (nV / √Hz)	2.9	8.0	4.0	18.0
$i_{n\ op}$ (pA / √Hz)	0.0069	0.003	1.5	0.010
$e_{n\ active\ comp}$	502 nV	1.39 µV	5.24 µV	3.12 µV
$e_{n\ total}$	2.99 µV	3.26 µV	5.47 µV	4.29 µV
(S+N)/N (dB)	84.46	83.72	79.23	81.33

Apart from the NE5534, these are all FET-input opamps and are characterized by negligible input noise current. The results obtained with the OPA134 opamp (Figure 1) are very good. Even more surprising is the respectable performance of the TL071. Being bipolar, the NE5534 is disadvantaged by its noise current and will give of its best in other configurations.

3. Influence of the transformer on the thermal noise

The eq. (1) can equally be expressed as:

$$R_a = n^2\cdot R_{mic}+n^2\cdot R_{p\ tr}+R_{sec\ tr} \quad (8)$$

It's clear that this sum comprises two terms, one for the microphone, the other for the transformer, for which an equivalent resistance can be defined, referred to the secondary:

$$R_{eq\ tranfo} = n^2\cdot R_{p\ tr}+R_{sec\ tr} \quad (9)$$

The contribution of the microphone and transformer to the thermal noise is then:

$$e_{n\ th\ mic\ et\ transf} = \sqrt{4\cdot k_b\cdot T\cdot\left(n^2\cdot R_{mic}+\text{R}_{eq\ transfo}\right)\cdot B} \quad (10)$$

The transformer multiplies the input voltage U_{mic} supplied by the microphone by a factor n. The calculation of the signal-to-noise ratio (S/N) at the secondary then gives:

$$\frac{S+N}{N} = \frac{n\cdot U_{mic}+\sqrt{4\cdot k_b\cdot T\cdot\left(n^2\cdot R_{mic}+\text{R}_{eq\ transfo}\right)\cdot B}}{\sqrt{4\cdot k_b\cdot T\cdot\left(n^2\cdot R_{mic}+\text{R}_{eq\ transfo}\right)\cdot B}} \quad (11)$$

By way of comparison, let us consider the same calculation for a transformer with no internal resistance:

$$R_{eq\ tranfo} = 0 \tag{12}$$

After simplifying, eq. (11) becomes:

$$\frac{S+N}{N} = \frac{U_{mic} + \sqrt{4 \cdot k_b \cdot T \cdot R_{mic} \cdot B}}{\sqrt{4 \cdot k_b \cdot T \cdot R_{mic} \cdot B}} \tag{13}$$

Let's compare equations (11) and (13): a transformer with no internal resistance would neither degrade nor improve the signal-to-noise ratio of the voltage supplied by the mic; in the case of a real transformer, these resistances can contribute substantially to increasing the noise. Hence the need for special attention in selecting this component.

Web Links and References

[1] www.analog.com

[2] www.ti.com

[3] www.buerklin.com

[4] www.lundahl.se/our-products/microphones/

[5] www.sowter.co.uk/pro-audio-transformers.php#

[6] www.jensen-transformers.com/mic_in.html

[7] *Mikrofonaufsätze* by Jörg WUTTKE, p. 83, in German (but worth taking the trouble to read!): www.schoeps.de/documents/Mikrofonbuch_komplett.pdf

[8] EN 61938 standard: Multimedia systems – Guide to the recommended characteristics of analogue interfaces to achieve interoperability.

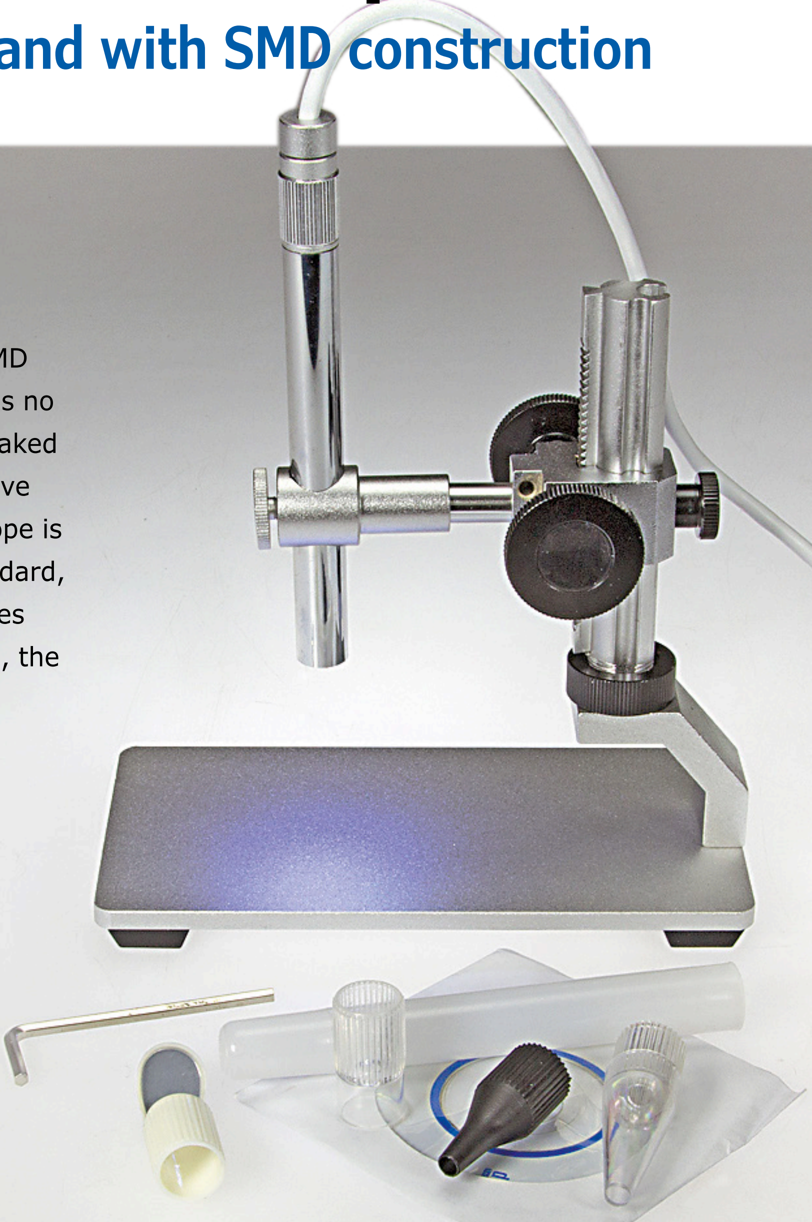
[9] Project pages: www.elektormagazine.com/140426 and www.elektormagazine.com/labs/1161

Review:

Andonstar USB microscope

A helping hand with SMD construction

The manual construction of prototypes with SMD components has become quite a challenge. It is no longer possible to inspect the result with the naked eye, since the components used these days have become too small. In this situation, a microscope is a very useful tool to have. Apart from the standard, optical microscopes, there are many USB scopes available nowadays. We've tested one of these, the Andonstar V160, in the Elektor Labs.



By **Luc Lemmens** (Elektor Labs)

A badly made solder connection, a solder bridge between two pins… it's difficult to avoid these unless you're using professional tools. An (illuminated) magnifying glass can help to spot faults like these, but it's often the case that the level of magnification is not enough to clearly see if there is something wrong. Another problem is that the magnifying glass ends up too close to the board to be able to work on the affected solder joint. There is a large selection of professional optical tools available to help you out with this, from high-quality stereo microscopes to inspection cameras and even X-ray imaging devices, but these all have the drawback that they come with a professional price tag as well. A USB microscope is an affordable alternative, although there are so many of them that it's difficult to pick the right one.

A sturdy stand is essential!

When you buy a budget USB microscope, the usability of the device depends less on the number of megapixels or the magni-

fication factor than you might expect. Instead, it's the mechanical construction that decides if the device earns its place on the workbench or if it should be chucked away with the rubbish! Most of the cheaper devices come with a wobbly stand, which causes the camera to lose focus as a result of the slightest vibration of the workbench, or when the camera is touched.
The Andonstar V160 is one of the few exceptions in that respect: this microscope has a robust stand that guarantees a stable image. Furthermore, everything can be set up precisely. The field of view can be adjusted using a set of knobs on the stand, and a rotating ring on the camera body is used to select the focus. The camera is nice and small and still shows a sharp image when it is several centimeters above a board, which leaves enough room for soldering underneath the microscope. The microscope can also be set at an angle in the stand, which lets you inspect components and solder joints from the side. An added advantage of this position is that it gives you more room for your tools when you work on the board.

For computer and tablet

USB microscopes are generally no different from ordinary USB cameras. They can therefore be used with most camera programs and are supported by all modern operating systems. The CD included with the V160 contains software for Windows only. On the Oasis Scientific website [1] we found an overview of programs that could be downloaded for most popular operating systems (apart from Linux!).
In the Lab, we've also connected the microscope to an Android tablet, which comes in very handy when there's not much room on the workbench. There are a number of apps available for this; we're using 'mScope', which is perfect! Your tablet would obviously need to support an USB OTG connection, and you'll need an USB-OTG adapter cable. All of the software packages mentioned have the facility to save photos or even movies onto the computer, which is very useful when documenting a project.

Built-in LED ring

The V160 comes with a LED-ring round the lens, with a controller in the connection cable to adjust the light intensity. It is very useful to have direct illumination of the object, but in some cases this can result in unwanted reflections in the image, particularly with a glossy (reflecting!) solder mask on the PCB. In those cases it's recommended to turn the LEDs off and use a (diffuse) light source next to the microscope to get the best possible image.

Indispensable tool

Since we've obtained this microscope, we've hardly used illuminated magnifying glasses for SMD work in the Elektor Labs, since the new microscope is much easier to work with. It does take some time to get the hand-eye coordination right, since you're used to looking at the tip of the soldering iron during solder work, rather than at a monitor. At the highest resolution (1600 × 1200 pixels) there is a small delay in the response of the image, which you also have to get used to. And anyway, it won't do it any harm to check that the soldering iron doesn't get too close to (or touch) the microscope. You'll soon get the hang of all this, and you'll find that this scope is something you don't want to be without when working with SMD components! ⏮

(160367)

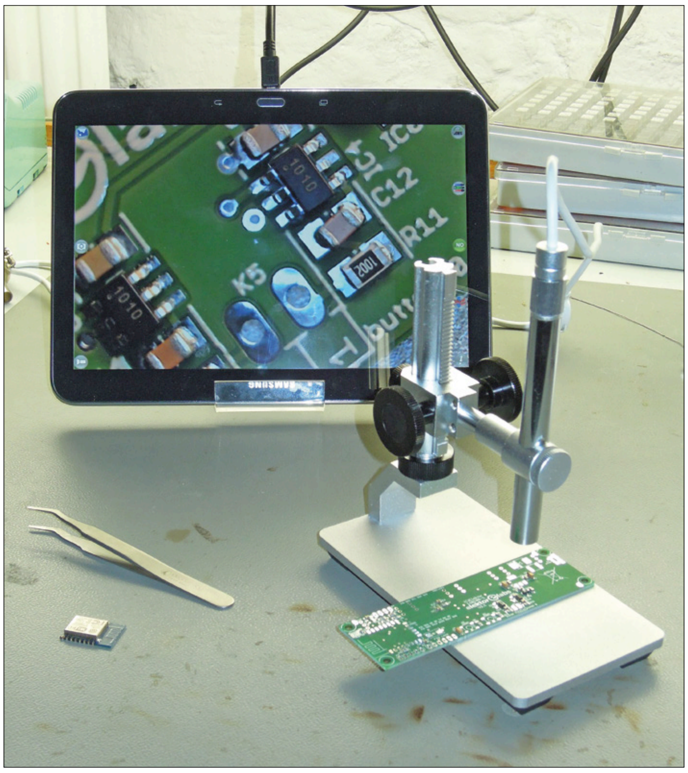


Figure 1. The Andonstar in use with an Android tablet.

A sturdy stand guarantees a stable image

Web Link

[1] www.oasisscientific.com/downloads.html

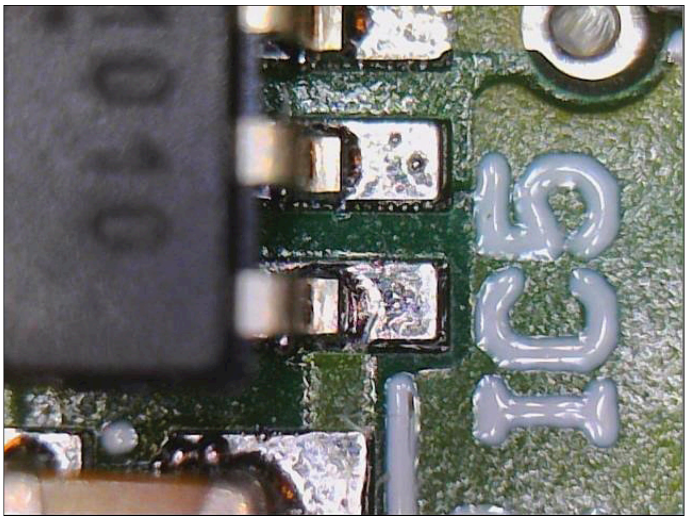


Figure2. Screenshot of a few 'feet' of an SMD IC on a PCB.

The I²C Bus

Part 1: The protocol

By **Josef Möllers** (Germany)

The inter-IC bus, or I²C bus for short, is a two-wire bus designed for connecting simple peripherals that do not require rapid transfer of large amounts of data. In this three-part series we will look at the essential features of the bus and see how it can be used in practice with a Raspberry Pi, Arduino or other system.

This short three-part series will not describe the I²C bus (also known, for example in the Atmel world, as the two-wire interface or TWI) down to the last detail, but instead aims to show how easy it is to build the bus into a design and how straightforward it is to use. In this first installment we will examine the data protocol; in the second installment we will test out the I²C bus on a range of systems from an Arduino and a Raspberry Pi to a full-fat PC; and then in the final part we will look at some typical I²C slave devices and give some ideas to show how it is possible to track down data transfer errors in a system.
The I²C bus was developed in the 1980s by Philips. If you would like to fully understand the I²C bus at the hardware level, rather than just as a black box under the control of ready-made software libraries, you will want to download the official specification from NXP [1] and commit to some serious reading.
A huge range of resources on the subject of the I²C bus is available on the Internet. One good example is [2], by the Hamburg-based consultancy Telos, which covers practically all aspects of communication on the bus.

A little light physics

The usual scenario is where a programmable microcontroller is used to control one or more peripheral components with fixed functions, such as a port expander, a real-time clock or an EEPROM. The microcontroller is the master, providing the bus clock (see below), while the peripheral devices are slaves. But it is also possible to connect two microcontrollers together using an I²C bus, with one of them taking on the

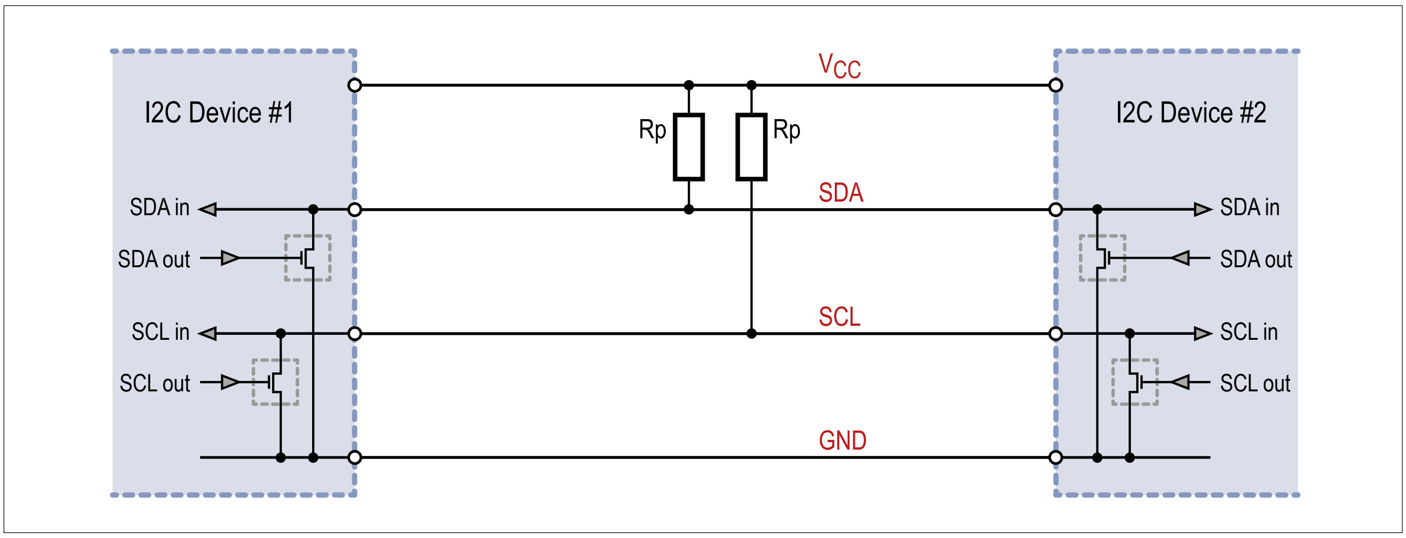


Figure 1. The I²C bus is based on a wired-AND connection.

role of the bus master. In the case where there are several slave devices, they are all connected to the same bus connections. An I²C bus can also have multiple master devices. The two wires that make up the I²C bus carry a clock signal (SCL) and a data signal (SDA). This allows the use of a simple four-way ribbon cable both to control a peripheral device and to provide it with power. To minimize crosstalk it is a good idea to route the ground connection between the two signal wires.

The signals on both wires operate with positive logic: a high voltage on SDA corresponds to a logic '1' and a clock on the SCL line consists of a brief positive-going pulse.

In the quiescent state a pair of pull-up resistors pull the two signals to a high level. The drivers in I²C devices have an open collector (or an open drain), which means that each of the two wires forms a bus-wide 'wired AND' connection (see **Figure 1**). The voltage level on one of the signal wires is only high when all the connected outputs are at a high level; if just one of the outputs is pulling low, then the whole wire will be at a low level. Each participant on the bus must therefore monitor the signal lines: if it wants to place a logic '1' on the bus it must make sure that no other participant is outputting a logic '0' at the same time. The value of the pull-up resistors is not especially critical, and should be in the region of a few kilo-ohms. The Raspberry Pi has these resistors already built in (1.8 kΩ pulling up to the 3.3 V supply). In the case of some microcontrollers such as the ATmega series there are pull-up resistors that can be enabled in software, but unfortunately they cannot be used for this purpose as their value (from 20 kΩ to 50 kΩ) is too high.

According to the specifications a high logic level should be at least 0.7 V_{DD} and a low logic level should be at most 0.3 V_{DD}. In practice, however, 5 V devices work with pull-up resistors to a 3.3-V supply, as in the case of the Raspberry Pi. However, out-of-tolerance signal levels can sometimes be the cause of problems, and in such cases level shifters will be required.

I²C slave modules with built-in pull-up resistors are also available. Normally, however, these should be removed or disabled, as if more than one slave has such resistors fitted they will be in parallel on the bus and the resulting overall pull-up resistance will be too low. An I²C bus should have just one pair of pull-up resistors, most conveniently located at or near the master device.

At this point we run into a problem: some devices that operate from a 3.3 V supply cannot tolerate higher voltages on their input pins. If the pull-up resistors are tied to a 5 V supply, these devices might give up the ghost. On the other hand, for the bus to operate reliably, we would like to stick to the specified ranges of voltage levels. One solution that allows 5-V and 3.3-V devices to share a common bus is to use a bidirectional level shifter, as described in [3]. The circuit uses two MOSFETs and some additional pull-up resistors to link the 5-V and 3.3-V sections of an I²C bus: see **Figure 2**.

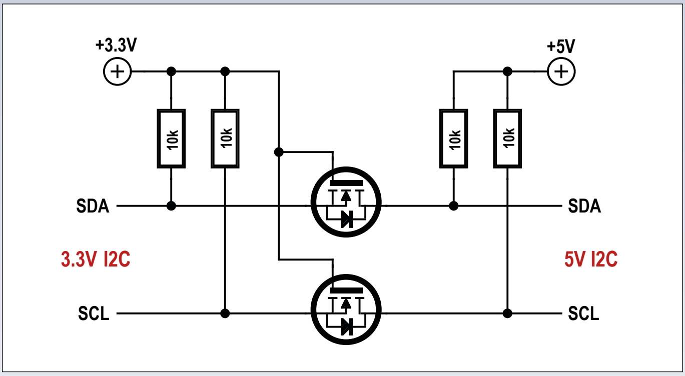


Figure 2. MOSFETs can be used to divide a bus into a 5 V section and a 3.3 V section.

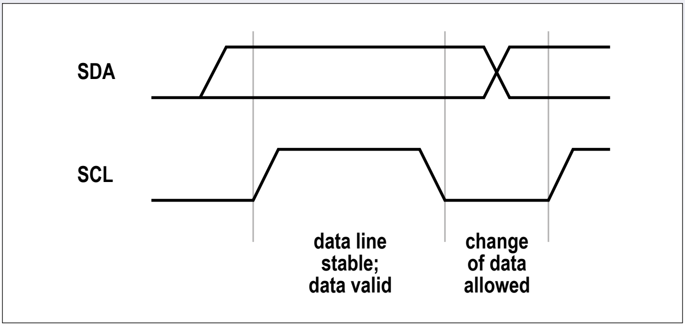


Figure 3. Timing of one data bit.

The protocol

The I²C bus is a pure master-slave bus: that means that communication is always initiated by the master, which addresses a slave and then exchanges data with it. A slave is not able to initiate communication by itself, as is possible for example in the case of a SCSI bus (using 'asynchronous event notification' and 'reselection'). As already noted above, a microcontroller can be programmed to take on the role of a master or a slave, and indeed can in theory change its role on the fly. However, in general, each bus participant has a fixed role as either a master or a slave.

The master is always responsible for setting the basic overall communication speed. Four speeds are officially supported:

- 100 kbps (standard mode or 'Sm', the default)
- 400 kbps (fast mode or 'Fm')
- 1 Mbps (fast mode plus or 'Fm+')
- 3.4 Mbps (high speed or 'Hs')

Data transfer is possible either from master to slave or from slave to master, but the clock signal, on the SCL line, is always generated by the master. Data bits are transferred synchronously to the clock, as shown in **Figure 3**: in order to send a bit, the transmitter must place the required logic level is on the SDA line, using positive logic. The master then releases the SCL line, which is then pulled high, and then shortly afterwards pulls it back low again. One bit has now been transferred and the transmitter can now proceed to the next bit. The SDA signal may not change state while SCL is high during a transmission.

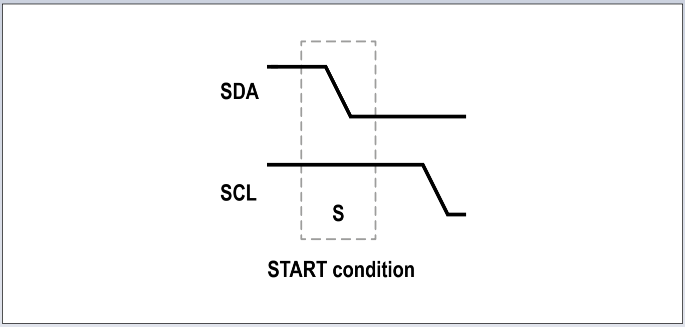


Figure 4. Start condition.

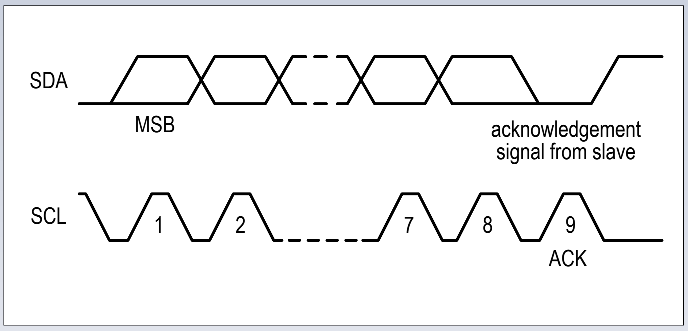


Figure 5. Sending the address bits.

1. To begin a transmission, the master pulls first the data line SDA and then the clock line SCL low (see **Figure 4**). This sequence is called a 'start condition'.

2. After the start condition the master first sends the slave address (see **Figure 5**), with most significant bit first. This is followed by the read/write bit: 0 here means that data will be transferred from master to slave and 1 means that data will be transferred from slave to master. The slave address is usually seven bits long (although the I²C standard also provides for ten-bit addressing, this feature is rarely used) and is hard-wired, either wholly or partly, by the manufacturer of the slave device. It is often possible to use external circuitry on a slave device to configure some of its address bits, which allows several instances of the same device to be connected to a single bus using different addresses for each. The slave address will be given in the device datasheet, or can be determined empirically using a program such as `i2cdetect`. The slave address plus the read/write bit makes a total of eight bits, or one byte.

3. After the address byte the master generates the SCL clock signal to transfer data, one byte at a time and with each byte transmitted most-significant bit first (see **Figure 6**). If the read/write bit in the address byte was 1, then the master expects the slave to place its data bits on the SDA line at the appropriate moment, before it generates the clock signal on the SCL line. If the read/write bit was a 0, however, the master itself places the data bits on the SDA line as well as generating the clock pulses. Each byte, including the byte comprising the address and the read/write bit, must be acknowledged by the receiver during the ninth bit time. Here a low level indicates a positive acknowledgement (ACK), while a high level is a negative acknowledgement (NACK). A NACK does not necessarily mean that something has gone wrong with the transmission: it is also used simply to indicate that the byte in question was the last one in the message. In the case of the address byte, a NACK means that no slave recognized the combination of address and read/write bit as valid.

4. At the end of a transmission the master releases first the SCL clock line and then the SDA data line: the pull-up resistors then pull the two signals to a high level. This sequence (see **Figure 7**) is called a 'stop condition'. To create the stop condition it may be necessary for the master to first pull the SDA line low.

The start condition and the stop condition are exceptions to the rule that SDA is not allowed to change while SCL is high. There is no limit to the number of bytes that can be transferred in a single transaction. Higher-level protocols, such as the Power Management Bus (PMBus) and System Management Bus (SMBus), which are based on the I²C bus, specify upper limits to the message length or define record formats that, for example, include a length field within the message. In any case 'unlimited' messages are not supported by all hardware implementations or by all software libraries, as in many instances the number of bytes to be received must be specified before a transaction can be started: one example of this is the 'Wire' library for the Arduino. The protocols also define timeout values (for example for clock stretching: see below) and data integrity checks ('Packet Error Checking', or PEC).
Each transaction, between a start condition and a stop condition, is always unidirectional: the master specifies the direction of data transfer for the whole transaction with the read/write bit.

Only a small number of devices have a sufficiently simple internal structure that they can be controlled using a single message. (One example is the PCF8574, which we will look at in the last part of this series.) Mostly I²C devices require the master to use a write transaction to set a parameter in the slave, such as a register pointer or a memory address, before a read transaction can be carried out. If the master wishes to send data to the slave, this can be done after the parameter within a single write transaction; however, if the master wishes to read from the slave, it has to start a new read transaction to do so. To avoid the situation where a second master interrupts this sequence and sends a different parameter to the same slave, the first master can omit the stop condition at the end of the parameter-setting message: this is called a 'repeated start condition'. The sequence is as follows:

```
start condition - address+write - register number -
   (repeated) start condition - address+read - data -
   ... - stop condition
```

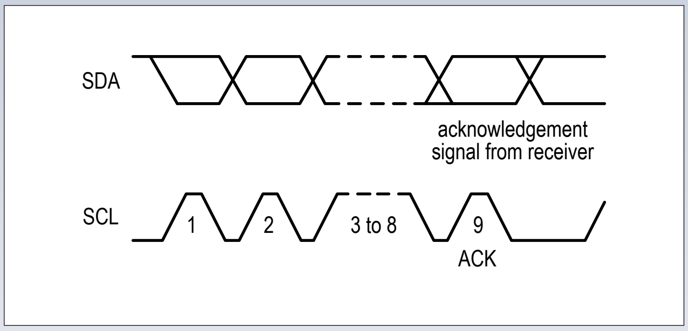


Figure 6. Sending the data bits.

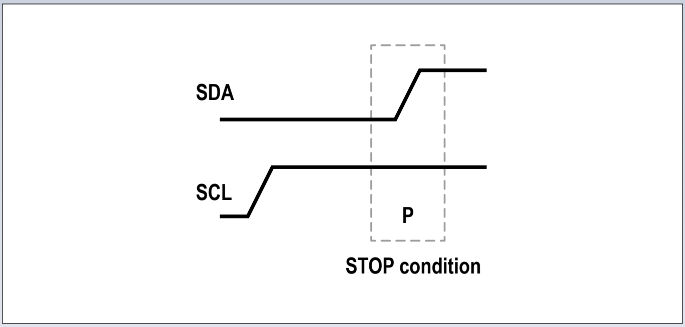


Figure 7. Stop condition.

Some peripheral devices such as memories or the RV-8523 real-time clock module automatically increment the register pointer or memory address after each byte is accessed; others, such as the LM75 temperature sensor, do not. In the former case it is possible to transfer the complete set of registers or the complete memory contents in one transaction. In the latter case, however, many separate transactions are needed; on the other hand, if the requirement is to read the same register again and again (which might be the case with the temperature register of the LM75) then there is no need to repeat the message to set the register pointer. Details on this point for a particular device will be found in its datasheet.

The I^{2}C protocol does not provide for interrupts. Of course a slave can use a separate signal to trigger an interrupt on the master device, which can then in turn interrogate the slave over its I^{2}C interface, but this does not form part of the standard I^{2}C protocol itself.

Clock stretching and arbitration

When a slave is asked to send data to the master, it will sometimes need a little time before it is ready. During this period the slave must find some way to stop or delay the master's clock signal. Since the bus signals are in a wired-AND arrangement, one way for the slave to do this is for it to hold the SCL line low. The master can detect this situation, as it is able to see that the SCL signal does not go high when it releases it. This mechanism is called 'clock stretching', although perhaps more accurately it might be called 'pause stretching' as it is the interval between clock pulses that is lengthened, not the length of the pulse itself.

As mentioned above, it is possible to have more than one master on a single I^{2}C bus. The bus is considered 'busy' in the period between a start condition and a stop condition, and no other master is permitted to generate a start condition during this period. A (would-be) bus master must therefore continuously monitor the bus to ensure that it only tries to send a start condition when the bus is free. If two masters should happen to send a start condition simultaneously, then it will in general be the case that either the slave addresses or at least the read/write bits will differ. That means that there will come a point where one master is trying to send a logic '1' while the other is trying to send a logic '0'. Because of the wired-AND arrangement, the master trying to send a logic '0' will see its logic '0' on the bus, but the master trying to send a logic '1' will not see its logic '1', and when it detects this it must immediately stop transmitting. The 'winning' master (the one that was trying to send logic '0') does not notice that anything has gone awry and continues its transmission.

The process described in the preceding paragraph is sometimes called (including in the official I^{2}C documentation) 'arbitration'. However, this is not really the correct term as there is no separate phase with priorities and timeouts: all that happens is that any master that wants to initiate a transaction waits for the bus to be free, starts transmitting, and watches what happens on the bus. There is nothing more to 'arbitration' as explained in NXP's documentation.

A master is allowed to terminate a transaction early at any time by sending a stop condition. A slave acting as a receiver can also terminate a transaction using a NACK, but a slave acting as a transmitter cannot. In fact, if a master is acting as a receiver and, as a result of a program bug or for some other reason either acknowledges a byte with a NACK or sends a stop condition, the transaction is not terminated. The slave may well stop sending data, but the master can carry on regardless, happily reading in whatever random data it happens to see on the SDA line.

Coming up

In the next installment in this series we will look at how various devices, from microcontrollers such as ATmegas and Arduinos to Raspberry Pis and PCs, can be used as I^{2}C masters and slaves. We will include software examples in C and in Python to show how to address a device, send commands to it and read data from it. ⏮

(160095)

Web Links

[1] www.nxp.com/documents/user_manual/UM10204.pdf

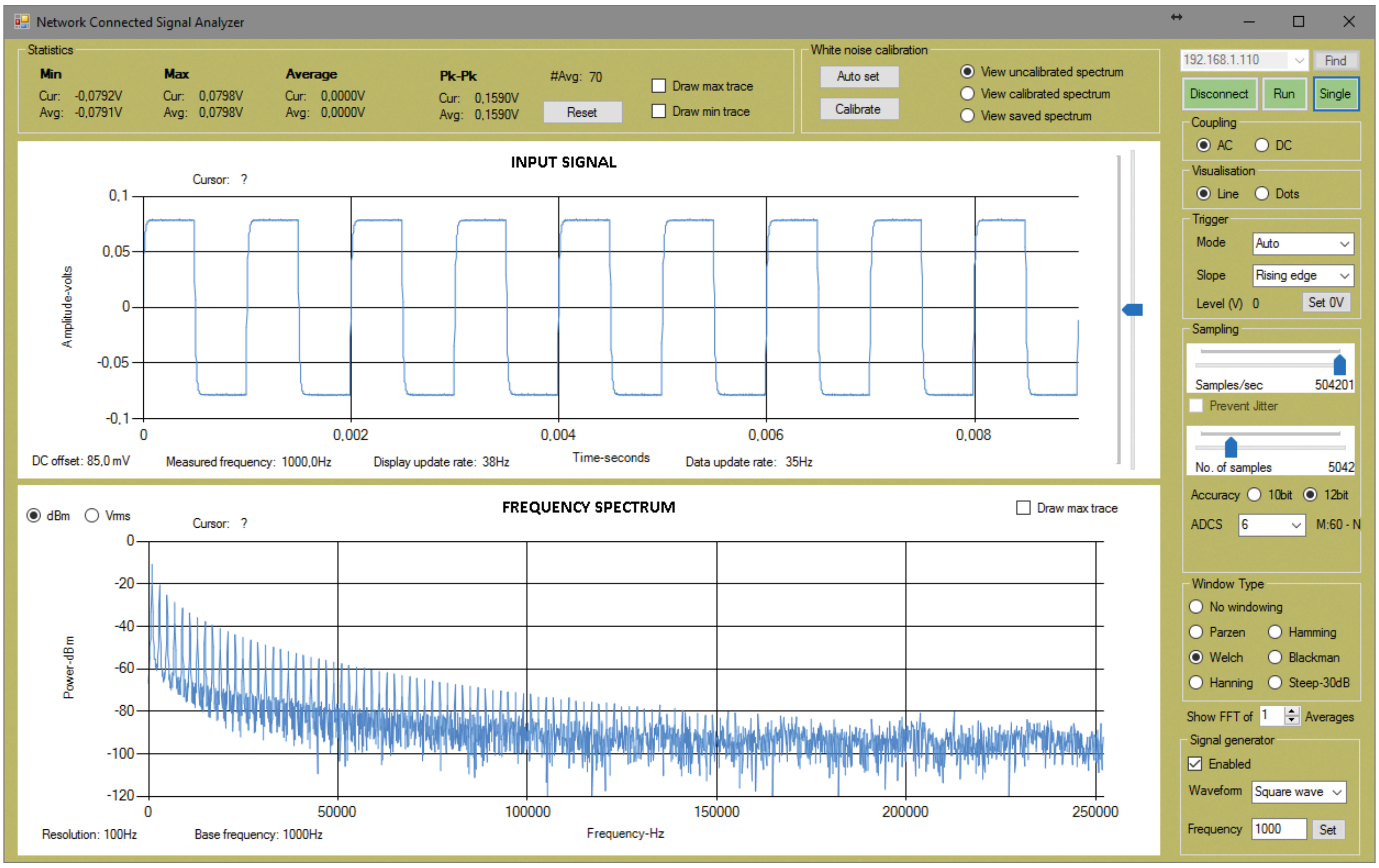
[2] www.i2c-bus.org

[3] http://playground.arduino.cc/Main/I2CBi-directionalLevelShifter

HOMELAB PROJECT

Network Connected Signal Analyzer Revisited

Faster, new features and even more easy to use


Network Connected Signal Analyzer

Statistics

Min	Max	Average	Pk-Pk
Cur: -0,0792V	Cur: 0,0798V	Cur: 0,0000V	Cur: 0,1590V
Avg: -0,0791V	Avg: 0,0798V	Avg: 0,0000V	Avg: 0,1590V

#Avg: 70

192.168.1.110

Coupling: [✓ AC] DC

Visualisation: [✓ Line] Dots

Trigger — Mode: Auto; Slope: Rising edge; Level (V): 0

Sampling — Samples/sec: 504201; No. of samples: 5042

Accuracy: 10bit [✓ 12bit]

ADCS: 6 M:60 - N

Window Type: No windowing, Parzen, Hamming, [✓ Welch], Blackman, Hanning, Steep-30dB

Show FFT of 1 Averages

Signal generator: [✓ Enabled]; Waveform: Square wave; Frequency: 1000

DC offset: 85,0 mV

Measured frequency: 1000,0Hz

Display update rate: 38Hz

Data update rate: 35Hz

Resolution: 100Hz

Base frequency: 1000Hz

[✓ dBm] Vrms

Figure 1. The new and improved network-connected signal analyzer user interface (UI).

In this article we present a software and firmware upgrade to the Network-Connected Signal Analyzer (NCSA) published in Elektor in March & April 2016. The upgrades represent significant improvements in connectivity, speed, functionality and ease-of-use while leaving the hardware unchanged. Please meet NCSA II.

By **Joost Breed** (Netherlands) and
Neal Martini (USA)

Let's briefly summarize the improvements in the four announced connectivity, speed, functionality and ease-of-use areas.

Improved connectivity

The NCSA II is plug-and-play when connected to a local area network (LAN). Also, two NCSA IIs can be connected to a LAN at the same time. This plug-and-play capability is a result of utilizing the Dynamic Host Configuration Protocol (DHCP) to automatically assign an IP address to the instrument.

Higher speed

The NCSA II runs several times faster than the original. The increased speed enables much faster oscilloscope and Fast

Fourier Transform (FFT) update rates. Multithreading in the PC client and Direct Memory Access (DMA) SPI in the dsPIC33 server are mainly responsible for the speed increases.

More functions

The oscilloscope in the signal analyzer has been extended and User Defined Functions (UDF) were added to the Synthetic Waveform Analyzer (SWA). The oscilloscope now features triggering, statistics generation and AC/DC coupling. The expanded SWA with UDF lets a user get the Fourier Transform of any continuous function that he/she mathematically describes. A very powerful tool!

Ease-of-use

is greatly improved in the NCSA II. The look and feel of the upgraded instrument now more closely resembles that of modern instrumentation. In general, things are more intuitive and consistent. This will become apparent after a few minutes using the new User Interface (UI).

The improvements in functionality and user experience can be had simply by upgrading to the new PC client software, but to get the connectivity and full performance improvements, the NCSA firmware needs to be upgraded too.

User interface walkthrough

Let's walk through the new user interface (UI) of the NCSA. **Figure 1** shows a typical example.

Connecting to the NCSA

When the analyzer is started, we do not know the IP address of the NCSA II. The 'Find' button in the top right corner will start searching for one or two NCSAs on the local network. The IP addresses of detected devices will be added to the dropdown list. After selecting the correct device, the 'Connect' button can be used to connect to the device.

Sampling

As with the previous version, the sampling parameters can be changed with the controls in the 'Sampling' panel. These parameters are samples/s, number of samples, the sample resolution (10 or 12 bits) and the ADCS (AD clock frequency divider). A 'Prevent Jitter' function has been added to automatically select the best sampling rate when moving the sampling rate slider. When this function is enabled, the sampling frequency is automatically changed to the nearest sampling frequency with the least jitter. The calculated sample frequency will be an integer multiple of the amount of cycles which are needed to do the A-to-D conversion divided by the desired frequency.

Triggering

The trigger system is done in software and the settings are similar to a regular oscilloscope. When the trigger voltage is set to a certain level (e.g. 1 V) and the slope is set to a rising edge, the algorithm will search through all the data received from the NCSA and collect the array index of each sample which is higher or equal to the trigger voltage and where the previous sample is lower than the trigger voltage (see **Table 1**).

Table 1. Trigger set at 1 V, rising edge. Trigger point found at position 1039 in the data array.

Index	1034	1035	1036	1037	1038	**1039**	1040	1041	1042
Voltage	0.800	0.820	0.883	0.950	0.969	1.003	1.012	1.133	1.306

This will result in a list of array indexes which all meet these criteria. The signal displayed in the graph will always start at the first trigger point found in the array. On the right side of the UI, in the 'Trigger' panel, it is possible to set a trigger level from −250 mV to +250 mV and also to set triggering on a rising or falling edge. It is also possible to choose either 'Auto' or 'Normal' triggering. In Normal mode the waveform will only be shown when the system has actually been triggered. This also means that when the system is not triggered, nothing will be displayed in the graphs. When triggering is set to Auto mode, the system will trigger just like in Normal mode, but if no trigger was seen for 200 ms or more, a single-shot capture will be done automatically. Auto mode is the default mode as is typically done in regular oscilloscopes. This allows the user to see what the signal looks like even when the trigger criteria are not met. Triggering can also be disabled by selecting trigger mode 'Off'. The trigger level can be set using the slider on the right side of the top graph. By pressing the 'Set 0V' button, the trigger level is reset to 0 V.
The signal analyzer defaults to 'Stop' mode. By clicking the red 'Stop' button the instrument enters 'Run' mode. The capturing can be stopped again by pressing the 'Run' button. It is also possible to capture a single-shot event. When the 'Single' button is pressed, the system will wait until it has been triggered and then switches to Stop mode immediately.

Coupling

In the 'Coupling' panel the desired signal coupling can be chosen. When set to 'DC', the signal is shown as it was measured. When set to 'AC', the DC value — calculated by the FFT — is subtracted from the measured signal before it is displayed.

Statistics

Various statistics like minimum, maximum, average and peak-peak voltage are calculated with each set of data received from the NSCA server. Averages are also measured and displayed; they can be reset by pressing the 'Reset' button. The statistics are displayed at the top of the UI. Selecting the checkboxes 'Show max trace' and 'Show min trace' will result in lines being displayed in the time domain graph. These lines represent respectively the maximum and minimum values captured from the time these checkboxes have been selected. These lines will be reset by either pressing the Reset button or disabling and re-enabling them again. The same functionality has been implemented for the maximum level of the FFT. At the bottom of the time domain graph the DC offset and the measured frequency are shown as well as the current update rate. The DC offset is collected from the FFT and the frequency is calculated from the time between two trigger points. The frequency domain graph also shows the base frequency which is the frequency of the FFT bin with the highest power.

Features

- Network-connected
- Plug & play
- Noise generator
- Oscilloscope
- Spectrum analyzer
- Subsampling engine

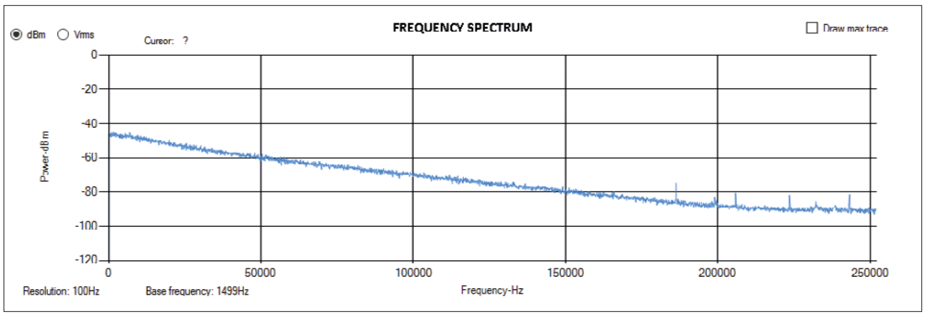


Figure 2. Input and output connected, not yet calibrated.

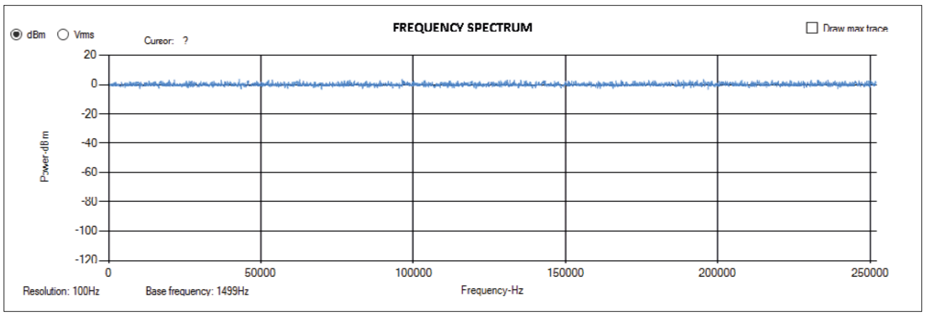
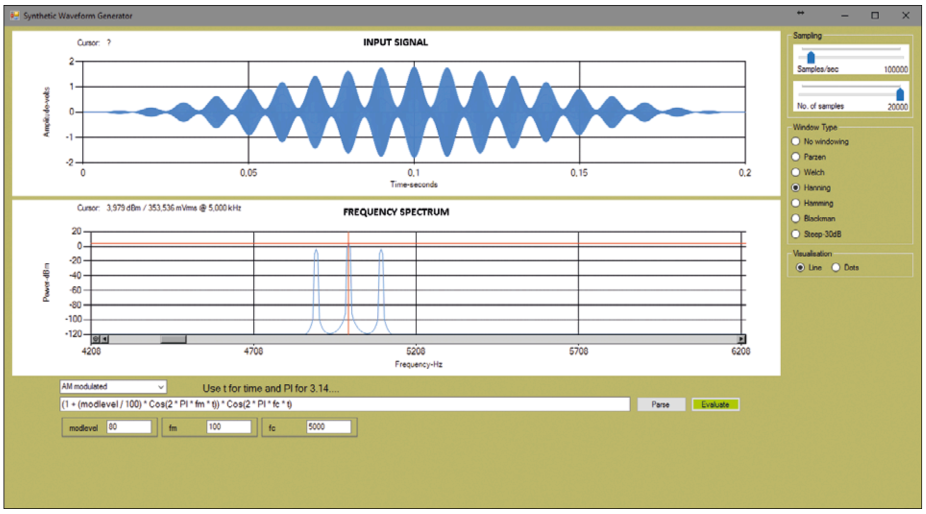


Figure 3. Input and output connected, calibrated.



AM modulated

Use t for time and PI for 3.14....

```
(1 + (modlevel / 100) * Cos(2 * PI * fm * t)) * Cos(2 * PI * fc * t)
```

modlevel 80 | fm 100 | fc 5000

Samples/sec 100000; No. of samples 20000; Window Type: [✓ Hanning]; Visualisation: [✓ Line]

Figure 4. Using the SWA user define capability to analyze an amplitude modulated (AM) signal. The parameter values for the modulation level, carrier- and modulation frequency can be entered in the input fields displayed.

Table 2. JP2 IP assignment jumper settings.

RB13 (JP2 pin 3)	RB14 (JP2 pin 1)	IP Assignment
GND	GND	192.168.1.123 fixed
GND	Open	192.168.0.123 fixed
open	GND	DHCP 1
open	open	DHCP 2

White noise calibration

One of the features of the NSCA II is its ability to measure the frequency response of a device such as a filter. This measurement can be done by using the instrument's internal noise generator as the signal source for the filter under test. The noise contains all frequencies in the spectrum. The filter will only let the frequencies through according to its frequency response. The output signal can be measured and the FFT shown is the frequency response of the filter.

Since the device contains a low pass filter on the output of the generator, the level of the generated noise is not equal for all frequencies. Therefore a calibration function is implemented to cancel out the characteristics of the instrument itself. The calibration is done by enabling the noise generator, connecting the instrument's input to its output, and then setting the FFT average to a high number such as 500. After 500 averages have been collected, the FFT is displayed. Then, by clicking the 'Calibrate' button at the top of the UI, the spectrum will be calibrated. From this moment on the captured FFT will act as a baseline for the measurements that follow (see **Figure 2** and **Figure 3**). Now the filter to be tested can be placed between the input and output of the instrument and its frequency response will be shown in the FFT graph.

The settings needed to start the calibration can be automatically set by clicking the 'Auto set' button. There is one important thing to note here. If the circuit under test does not have a 50 Ω input impedance, a dummy 50 Ω shunt needs to be inserted. This is required so that the noise generator sees the same impedance when connected to the NCSA or to the circuit under test. Also, since the input to the NCSA is DC-coupled, a capacitor should be used to AC couple the circuit under test to the DC biasing at the NCSA input.

Synthetic waveform analyzer

The Synthetic Waveform Analyzer (SWA) has been upgraded to enable the user to enter and evaluate custom formulas sampled with the selected sample frequency, number of samples and FFT window. The SWA uses an open source library called NCalc for parsing the entered formula. After entering the formula and hitting the 'Parse' button, the unknown variables are displayed as input fields below the formula box. Here the user can enter the desired values (**Figure 4**). By clicking the 'Evaluate' button, the entered variable values are taken from the input fields and the formula is evaluated for each sample. The result is displayed in the time domain graph. The FFT of the signal is also calculated and displayed in the bottom graph.

Plug & play connectivity

To connect with the local area network (LAN), the original NCSA user had a choice of four fixed IP addresses, selected via connector JP2. The new firmware uses the Dynamic Host Configuration Protocol (DHCP) instead to automatically get an IP address assigned to it when it is plugged into a LAN. Additionally, it is now possible to have two NCSAs simultaneously connected to a single LAN.

DHCP IP assignment

Remember, ultimately, we are trying to make a connection between a PC (client) and the NCSA (server). DHCP is the first step in establishing this connection. A DHCP server controls the IP assignment process and is typically located in the LAN router. To help visualize the phases of DHCP, we are utilizing the NCSA's debug feature. Debugging is accomplished by placing a serial-to-USB adapter between a PC and the NCSA's debug connector K6. Then, using a terminal emulator (e.g. Tera Term), you can view the various steps in the DHCP process (**Figure 5**). Connector JP2 is used to select one of four IP choices, see **Table 2**. The first two entries allow for two fixed IP address, if desired, similar to the original NCSA. The second two choices allow for one of two DHCP assigned IP addresses to be selected. Two choices are here so that we can have up to two NCSA's on the same LAN.

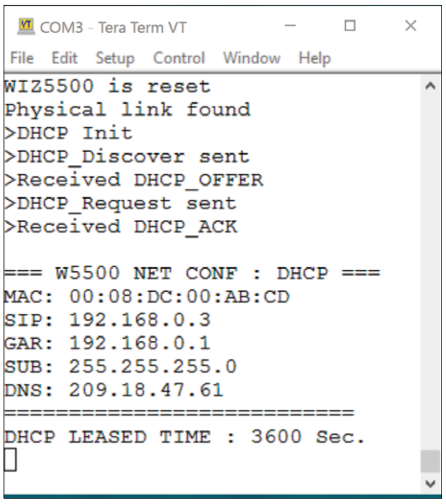
At this point, all we have is an IP address assigned to the NCSA by the LAN router, an IP address that is unknown to the client software running on the PC. How can the PC client discover the NCSA's IP address? This problem is more involved than you might think.

Discovering the NCSA

The file called NetworkScanner.cs in the C# project contains the code that allows the PC to find the NCSA II's IP address. Let's go through the various steps in the process and show a few code snippets from NetworkScanner.cs that are key to each step:

- **1.** (**Listing 1**) Find the IP address of the LAN gateways (typically in the router), and check that they are up and running. If they are present and running, get the gateway IP address. For example, a typical router might have a gateway address 192.168.0.1.

- **2.** (**Listing 2**) Look for the presence of an NCSA II with an IP address that falls in the range of the network gateway. This is done by pinging all the possible addresses and waiting for a successful reply. For example, if the gateway address is found to 192.168.0.1, IP addresses 192.168.0.2 through 192.168.0.255 are pinged. This would be very time intensive if done one at a time, so multithreading is used to run concurrent ping threads. When a ping is completed it will fire the PingCompleted Method associated with the original ping.



```
WIZ5500 is reset
Physical link found
>DHCP Init
>DHCP_Discover sent
>Received DHCP_OFFER
>DHCP_Request sent
>Received DHCP_ACK

=== W5500 NET CONF : DHCP ===
MAC: 00:08:DC:00:AB:CD
SIP: 192.168.0.3
GAR: 192.168.0.1
SUB: 255.255.255.0
DNS: 209.18.47.61
===========================
DHCP LEASED TIME : 3600 Sec.
```

Figure 5. NCSA debug output clearly shows the DHCP phases.

Listing 2. Look for the presence of an NCSA.

```
private static void Ping(IPAddress host, int attempts, int timeout)
{
  ...
  Ping ping = new Ping();
  ping.PingCompleted += PingCompleted;
  ping.SendAsync(host, timeout, host);
  ...
}
```

Listing 1. Find the IP address of the gateway.

```
static IPAddress NetworkGateway()
{
  ...
  foreach (NetworkInterface f in NetworkInterface.GetAllNetworkInterfaces())
  {
    if (f.OperationalStatus == OperationalStatus.Up)
    {
      foreach (GatewayIPAddressInformation d in f.GetIPProperties().GatewayAddresses)
      {
        if (d.Address.AddressFamily == AddressFamily.InterNetwork)
          ip = d.Address;
}  }  }  }
```

- **3.** (**Listing 3**) When a IP ping is completed, check to see if there was a successful response. If it was successful, try and establish a TCP/IP connection using that IP address and Port 4000 (hard coded into the NCSA II TCP/IP code). If the TCP/IP connection is successful, go get the NCSA II's Media Access Control (MAC) address.

- **4.** (**Listing 4**) To get the hardware (MAC) address of the NCSA, we need to look in the PC's Address Resolution Protocol (ARP) table. This table contains a listing of the IP addresses and corresponding MAC addresses that the PC knows are connected to the LAN.

Every time a successful TCP/IP connection is made, the PC adds an IP/MAC address pair to its ARP table. So, if we query the ARP table, we can find the NCSA's MAC address (**Figure 6**).
Incidentally, the MAC address allows us to have two NCSAs on a single LAN. The NCSA changes the most significant field of the W5500's default MAC address depending on the settings of JP2. The PC client sees two different MAC addresses so it knows there are two different IP addresses to choose from. In fact, if you want to get clever, you could add a third NCSA by selecting a fixed IP address for the third device.
That's it. The NCSA has an IP assigned to it, and the PC knows what IP it can use to communicate with it. This will greatly facilitate the process of getting the NCSA up and running on your LAN.

Performance enhancements

The improvements in the NCSA speed are primarily a result of the application of multithreading in the client software, and the utilization of direct memory access in the dsPIC33-based server. Let's briefly discuss each of these.

Multithreading

When writing a PC program without taking multithreading into account, all code will be executed on one and the same thread, the main thread, which is created when the program is started by the OS. That thread will be executed on one of the logical processors of the CPU. A PC with a quad-core processor and hyper threading enabled has eight logical processors. This means that only 12.5% of the CPU is utilized when that logical processor is running at 100%. Multithreading gives the application access to more of the CPU's processing potential.
In the first version of the client software all work was done on a single thread in the following order: requesting data from the NCSA, receiving data from the NCSA, converting volts into dB, calculating the FFT, displaying the time graph and displaying the FFT graph. When one or more of those functions take a long time, the overall update rate will slow down because they are all executed sequentially. Another issue is that the main thread is also used to handle the UI. So when the thread is busy doing other stuff, the UI can react very slowly on user input such as pressing a button. Rendering the UI will be slow too. The solution to speed things up is to use a separate thread for each main function which distributes the load to more logical processors.
In the new client software requesting and receiving of captured data is done in a separate thread. Also the FFT calculations are done on their own thread. That way more is being done in parallel and the performance increases significantly. The display update rate rises and the UI reacts much faster on user input. When the graphs are being rendered by the main thread, the other

Table 3. Update rates compared with and without the optimizations (f_s = 504,201 Hz).

N	NCSA update rate (Hz)	NCSA II update rate (Hz)
5,000	8.33	42
10,000	3.96	22
15,000	2.48	14
20,000	1.70	11

Listing 3. Did an NCSA respond?

```
private static void PingCompleted(object sender,
PingCompletedEventArgs e)
{
  ...
  if (e.Reply != null && e.Reply.Status == IPStatus.Success)
  {
    // Try to connect to the port
    if (TryConnect(ip, _port))
    {
      // Can connect.
      string macaddres = GetMacAddress(ip);
      ...
}  }  }
```

Listing 4. Use ARP table to get MAC from IP address.

```
public static string GetMacAddress(IPAddress ip)
{
  ...
  if (SendARP(BitConverter.ToInt32(ip.GetAddressBytes(),0),0,macAddr,ref macAddrLen) != 0)
    return "00:00:00:00:00:00";
  ...
}
```

threads are already receiving the next samples and doing the FFT calculations. This resulted in a display update rate of about 40 Hz instead of 8 Hz when acquiring 5042 samples at 504,021 samples/s, an increase of 5×! (**Table 3**).

Direct memory access

In the original NCSA firmware, the dsPIC33's SPI peripheral is used by the wiznet_read() and wiznet_write() functions in the W5500.c ioLibrary to control the transfer of data to and from the W5500 network chip. The transfer speed was limited by the number of instructions needed by the library functions because they have to be executed by the dsPIC's CPU. The new firmware uses the dsPIC33's direct memory access (DMA) capabilities to significantly improve data transfer rates as it allows transferring data between memory and the SPI peripheral without using the dsPIC's CPU. The byte transfer rate has almost doubled: 1.3 MB/s versus the original 700 KB/s rate.

As it turns out, utilizing the DMA SPI in the dsPIC33 is not that straight forward. Two DMA channels are required to make things operate properly. DMA channel 1 sends bytes to the W5500, and DMA channel 2 does a dummy read after each byte transfer, as required by the dsPIC33's DMA SPI system. Both DMAs are in one shot mode, which means the DMA controller transfers a block of data, generates an interrupt when done, and then stops until another block transfer request arrives. **Listing 5** shows the essence of the DMA SPI control.

The need to do a dummy read after each byte write is not ideal. Also, having to manually force the first byte transfer to get things going is awkward. It would have been much easier to only use a single DMA channel, start it, and wait for the transfer to complete. But, unfortunately, that is not the way it is designed in the dsPIC33.

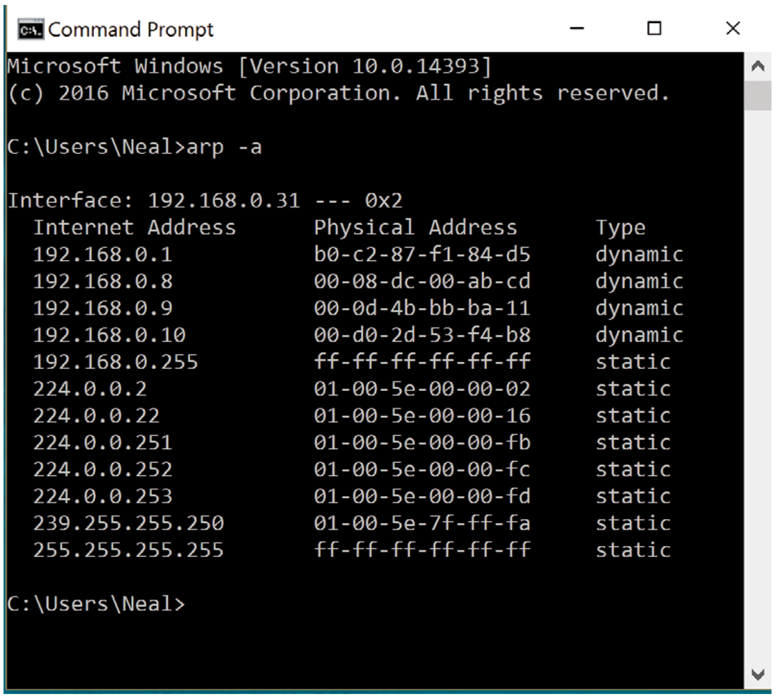
Upgrading the NCSA

The original NCSA will operate with the new PC client software, and the new UI and the enhanced SWA with UDF will be accessible. However, the DHCP plug-and-play network configuration and some of the speed enhancements won't be available unless you upgrade the dsPIC33's firmware.

To upgrade the Visual Studio client code, you need to download Visual Studio 2015

Listing 5. SPI with DMA on a dsPIC33.

```
done = false;  //using dma and spi for faster uP to W5500 transfers
SPI1STATbits.SPIEN=0; //keep SPI off until ready for transfer

//Set up DMA 1 to transfer from RAM at pBuf to SPI
DMA1CNT = len-1;  //block size
DMA1STAL = (unsigned int) pBuf;  //point to data buffer
IFS0bits.DMA1IF = 0;  //clear DMA interrupt flag
IEC0bits.DMA1IE = 1;  // Enable DMA interrupt
DMA1CONbits.CHEN = 1;  //turn DMA1 on

//Set up DMA 2 to do dummy reads after each write;
//required for dsPIC33 DMA SPI
DMA2STAL = (unsigned int) &dummy;
DMA2CNT = len-1;
IFS1bits.DMA2IF = 0;
IEC1bits.DMA2IE = 1;
DMA2CONbits.CHEN=1;

//force the first byte transfer after enabling SPI;
//required by dsPIC33 to start process
SPI1STATbits.SPIEN=1;//turn on SPI
DMA1REQbits.FORCE=1;
while (DMA1REQbits.FORCE==1);

//wait for DMA transfer to complete; done set true by DMA interrupt
while (done==false);
```



```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Neal>arp -a

Interface: 192.168.0.31 --- 0x2
  Internet Address      Physical Address      Type
  192.168.0.1           b0-c2-87-f1-84-d5     dynamic
  192.168.0.8           00-08-dc-00-ab-cd     dynamic
  192.168.0.9           00-0d-4b-bb-ba-11     dynamic
  192.168.0.10          00-d0-2d-53-f4-b8     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.2             01-00-5e-00-00-02     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  224.0.0.253           01-00-5e-00-00-fd     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

C:\Users\Neal>
```

Figure 6. The ARP table showing the NCSA on the LAN (type `arp -a` in a command prompt window to display such a table). The entry for IP = 192.168.0.8 belongs to our NCSA with the W5500 default MAC address 00-08-dc-00-ab-cd.

Generate a noise-calibrated frequency response

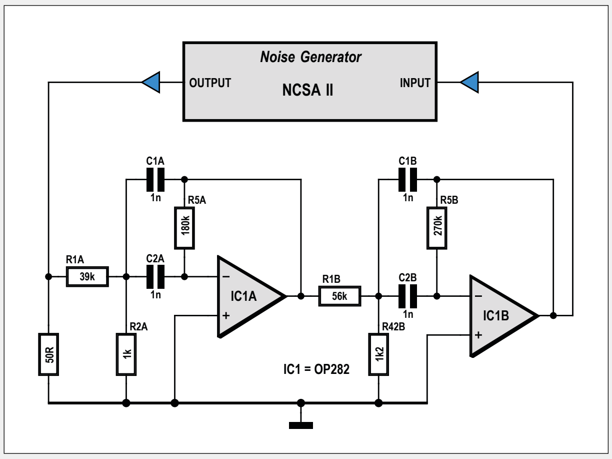


Figure 7. Analyzing an active band-pass filter using the NCSA's internal noise generator.

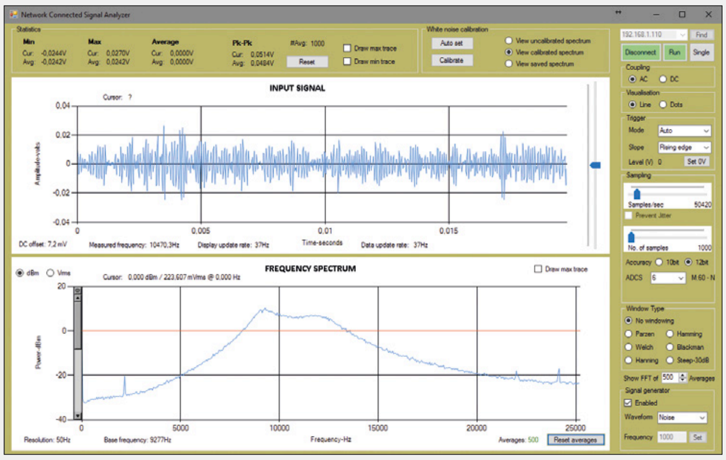


Figure 8. The response of the filter from Figure 7 obtained by using the noise calibration feature.

Let's use the NCSA II's internal noise generator to analyze the frequency response of the simple active band pass filter (BPF) shown in **Figure 7**. First, the NCSA II captures the internal noise generator's direct connect spectrum as a reference, then, relative to this reference, and with the noise generator now connected to the filter input, the UI displays the filter's frequency response (**Figure 8**). Examining the spectrum plot one can clearly see that the BPF's center frequency is at 10 kHz, has a bandwidth of about 5 kHz with a roll-off rate of approximately 24 dB/octave, as expected for a four pole BPF.

Subsampling mode

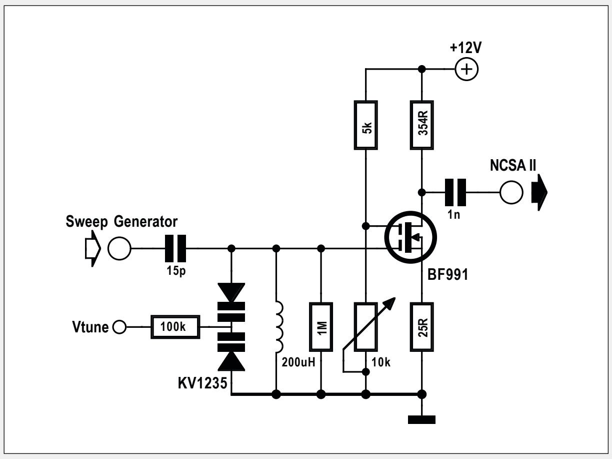


Figure 9. Tunable AM radio FET-based LC bandpass filter.

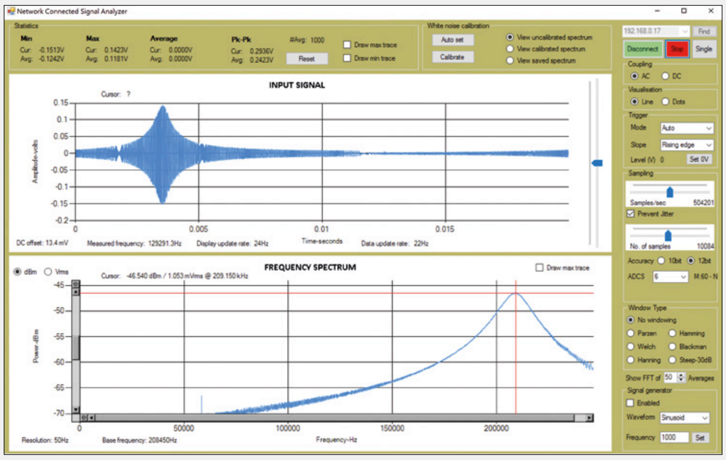


Figure 10. Swept frequency response of the high-frequency BPF. The actual filter is centered at 1.201 MHz. Subsampling is used to view this high frequency.

The next application is another BPF, only this time the filter has a much higher center frequency (1.2 MHz), see **Figure 9**. The circuit is an FET tuned LC BPF that is designed to be used as an AM radio front end. The filter's center frequency is tuned by varying the DC voltage Vtune, which, in turn, varies the capacitance of two varactor diodes in parallel with an inductor. The filter's tuning range is from about 580 kHz to 1.6 MHz. Since the instrument's sampling rate cannot meet the normal Nyquist criteria of twice the highest frequency, we have to utilize the NCSA II's subsampling operating mode [2]. An external signal generator is used to sweep the filter's input signal from 1 MHz to 1.5 MHz in 10 ms. The filter is tuned to about 1.2 MHz. **Figure 10** shows the filter's response.

You can see in the upper plot the filter's output growing as the input signal frequency passes through the filter's center frequency. In the spectrum plot you see the filter's bandpass response peak at 209,150 Hz. Recall that in subsampling the actual frequency can be calculated by taking into account the multiples of the sampling frequency. Here we get the filter's exact center frequency f_c = 1,201,552 Hz (209,150 + 2 × 504,201 Hz).

Analyze a QAM modulator

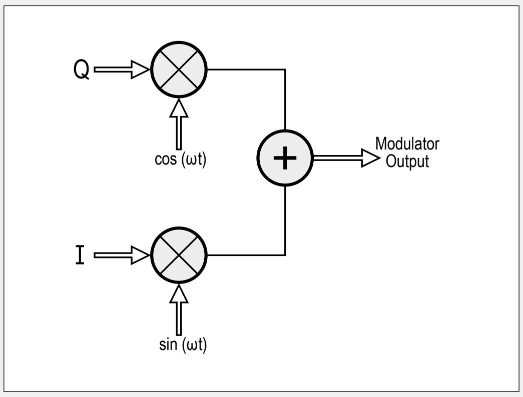
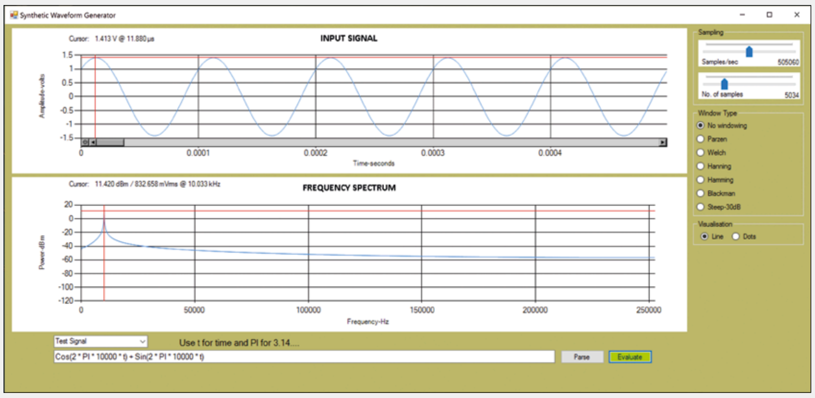


Figure 11. IQ modulator block diagram.



Cursor: 1.413 V @ 11.880 µs

Cursor: 11.420 dBm / 832.658 mVrms @ 10.033 kHz

Samples/sec 505060

No. of samples 5034

Window Type: [✓ No windowing] Parzen Welch Hanning Hamming Blackman Steep-30dB

Visualisation: [✓ Line] Dots

Use t for time and PI for 3.14....

Cos(2 * PI * 10000 * t) + Sin(2 * PI * 10000 * t)

Figure 12. The NCSA's synthetic generator IQ modulator output.

This example demonstrates the NCSA's synthetic waveform analyzer's (SWA) user defined function (UDF) capability. Here we analyze a classic QAM modulator (also called IQ modulator) which is used extensively in many of today's digital communication systems. **Figure 11** shows a typical IQ modulator configuration. In cable TV, for example, 256-QAM is often used, in which I and Q can each take on 16 distinct amplitudes. The equation describing the IQ modulator is:

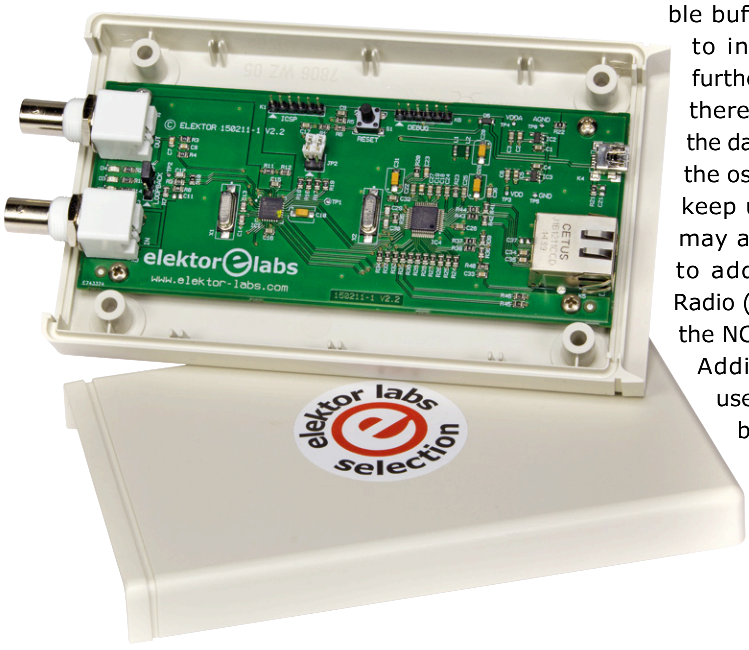
$$Q \cos(\omega t) + I \sin(\omega t) = C \cos(\omega t + \theta)$$

where $C = \sqrt{(Q^2 + I^2)}$ and $\theta = \tan^{-1}(Q / I)$

Examining the equation, we see that for a set of I and Q values, the modulator's output will be a cosine wave with amplitude and phase given in the equation. Let's put this function into the SWA and see if our results match the theory.

In this example I = Q = 1 volt. Using the equation, the resulting waveform should be a cosine with a starting phase at t = 0 of 45° and an amplitude of 1.414 V. The SWA's output plots shown in **Figure 12** match these predicted values. The user can change the values of I and Q in the entry boxes and see the resulting waveforms.

—available free from Microsoft — and download the NCSA Visual Studio project from [3]. After building the project, you are ready to go.
To upgrade the original NCSA, you need to download the MPLABS IDE — available free from Microchip — and download the MPLABS.X project from [3]. After compiling the code, you will need a programmer to load the firmware into the dsPIC33. The inexpensive PICkit 3 programmer (available from Microchip) that plugs directly into the ICSP connector K1 can be used for this. Those who do not want to invest in a programming pod or who are afraid of doing something wrong can use the Elektor reprogramming service (see Shopping list).



What next?

There are still a lot of possibilities for adding capabilities to the NCSA. Double buffering can be added to increase speed even further to the point where there are no time gaps in the data flow. This will allow the oscilloscope and FFT to keep up with real time. It may also be possible then to add Software Defined Radio (SDR) functionality to the NCSA.
Additionally, the SWA's user defined feature can be augmented with the addition of nonlinear function capability. If nonlinear functions like Rect, Sign, delta, etc., were added, the analysis of very broad communication applications could be analyzed.
All and all, we have a very powerful, evolving analysis tool. Stay tuned! ◄

(160362)

Web Links

[1] NCSA Part 1:
www.elektormagazine.com/150211

[2] NCSA Part 2:
www.elektormagazine.com/150694

[3] www.elektormagazine.com/160362

FROM THE STORE

→150211-91
NCSA ready built module

→150211-71
Case for NCSA

→160362-41
Reprogramming NCSA to NCSA II

e LABS PROJECT

4-Channel, 2-Flavor

with XBee or nRF24 radio modules

By **Somnath Bera** (India) & **Roy Aarts** (Elektor Labs)

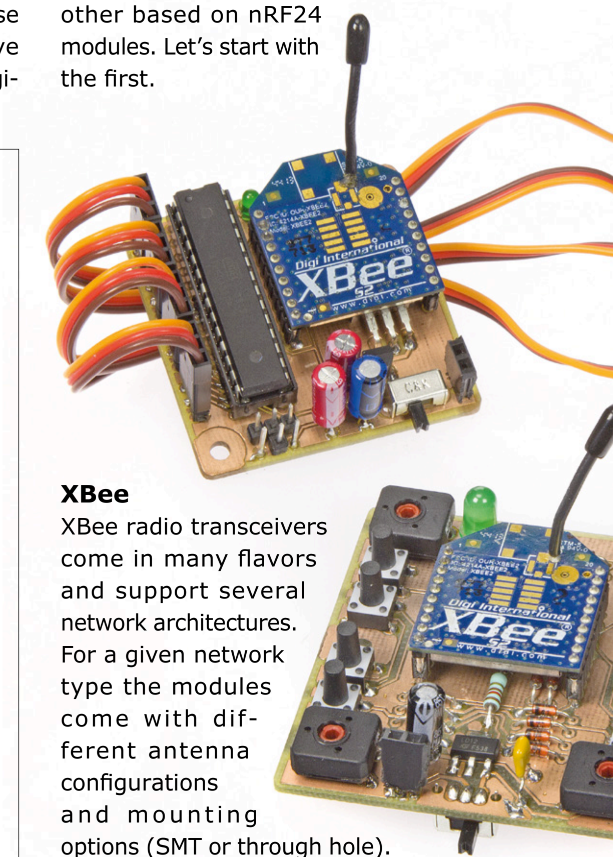
Remote Control

Features

- Four channels
- Two architectures
- Long range
- Low cost

Cheap ISM-band radio modules are great for making simple remote-controlled (RC) toys and other RC devices which need nothing more than throwing some switches on and off and running a couple of motors. However, when a greater degree of control is needed, for instance when running a few servos for controlling a model plane's rudder, elevator, aileron, and a brushless motor while at the same time receiving feedback from each of these actions, your plain simple radio control is, ermm … too simple.

In view of its complexity, the above situation calls for a more elaborated radio system. The popular XBee modules from Digi International are just that. Initially designed for point-to-point and star communications at over-the-air baud rates of 250 Kbit/s, they have evolved into a family of easy-to-use wireless transceiver modules covering several network architectures including cellular, ZigBee, Thread, 802.15.4 and Wi-Fi. In this article we will use the XBee S2C (formerly ZB) modules running a ZigBee protocol stack.
Although XBee modules are easy to use indeed, they can be on the expensive side, which is why our Elektor Labs engineer ventured out to achieve a similar result by using ultra-cheap nRF24 radio modules instead. This article therefore presents two four-channel remote controls, one using XBee modules and the other based on nRF24 modules. Let's start with the first.



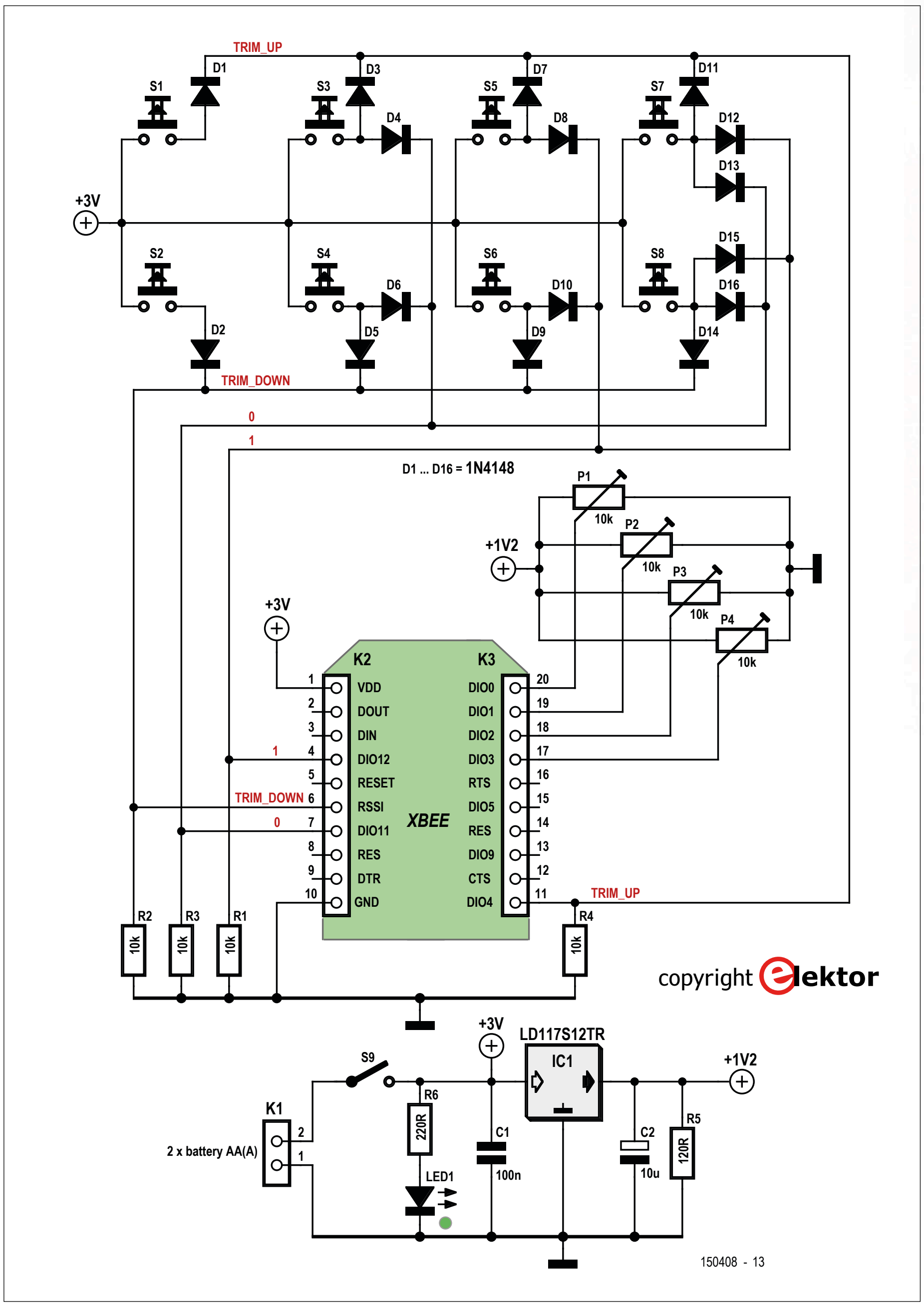


Figure 1. Schematic of the XBee transmitter.

XBee

XBee radio transceivers come in many flavors and support several network architectures. For a given network type the modules come with different antenna configurations and mounting options (SMT or through hole). The ZigBee modules furthermore exist in two versions: normal and Pro, where the Pro offers higher range. The normal version has a range of 60 meters (approx. 200 feet) in indoor and urban situations with a line-of-sight (LOS) range of about 1,200 m (approx. 3,500 feet). The XBee-Pro has a 90 m indoor/urban range and an LOS range of about 3.2 km (2 miles). With a Yagi directional antenna the Pro can achieve LOS ranges as far as 20 km (12 miles). Note that the XBee-Pro is not certified for use in Europe.
The modules are roughly the size of a 2-euro coin (had it been square), that's 0.96 × 1.09 inches (24 × 28 mm), the Pro being slightly longer than the regular version. Not only is the Pro somewhat larger, it also consumes more power. The modules have 20 pins divided over

two 10-way headers, 15 of which can be used as digital I/O and four as analog inputs. As said before, several antenna configurations are possible: the through-hole modules exist in chip antenna, wire, U.FL, RPSMA versions. We opted for the simple wire antenna type which is enough to get you going with all sorts of XBee experiments.

The ZigBee modules are intended for mesh networking and routers, and other network equipment exist to create big networks covering large areas. However, point-to-point networking as we will be doing here is possible too.

The XBee transmitter

Figure 1 shows the schematic of the XBee-based transmitter. Since in this case the module can work alone thanks to its pre-programmed embedded 32-bit ARM Cortex M3 processor, an external microcontroller is not needed. The XBee firmware scans its analog and digital inputs all by itself and transmits the values to an 'interested' receiver over a virtual (wireless) serial link.

Four of the module's I/O pins are used for reading voltages. We used trimmers but when you want to control a plane or a car it may be better to replace them by two joysticks. The horizontal axis of each joystick can be connected in place of P1 or P3, the vertical axis replaces P2 or P4. Replacing a trimmer by an LDR with series resistor or some other sensor that produces a voltage is possible too. The analog inputs can read voltages up to 1.2 V. The trimmers are supplied by a 1.2-V voltage regulator IC1 so that this voltage will be independent of the battery voltage.

The idea here is to control servos. However, in many cases servos need a small constant offset to compensate for a mechanical imprecision and pushbuttons S1 to S8 can be used for that. The odd-numbered pushbuttons move the offset up while the even-numbered ones decrease the offset. Pushbutton pair S1-S2 controls channel 1; pair S3-S4) is for channel 2; pair S4-S5 handles channel 3 and we leave it to you to guess which channel is controlled by pair S7-S8. Sixteen diodes, D1 through D16 are used to encode nine pushbutton possibilities. That's nine because 'all keys released' is a valid and important position too. Multiple keys pressed at the same time are rejected by the receiver.

That concludes the description of the transmitter hardware — how to program it will be described later. Good news: it does not require learning a programming language first.

The XBee receiver

The schematic of the receiver (**Figure 2**) differs from the transmitter because now a microcontroller is needed to decode the serial data received from the transmitter. The XBee module's default firmware cannot do this automatically since it doesn't have any knowledge about the meaning of the data. Instead it will simply spit out

PROJECT INFORMATION

Modeling
XBee nRF24 Arduino
Remote Control RC

➡ entry level
intermediate level
expert level

2 hours approximately

Computer, Soldering iron
USB-to-Serial adapter
AVR ISP programmer

XBee: €100 / £85 / $105
nRF24: €30 / £25 / $32
(costs approx.)

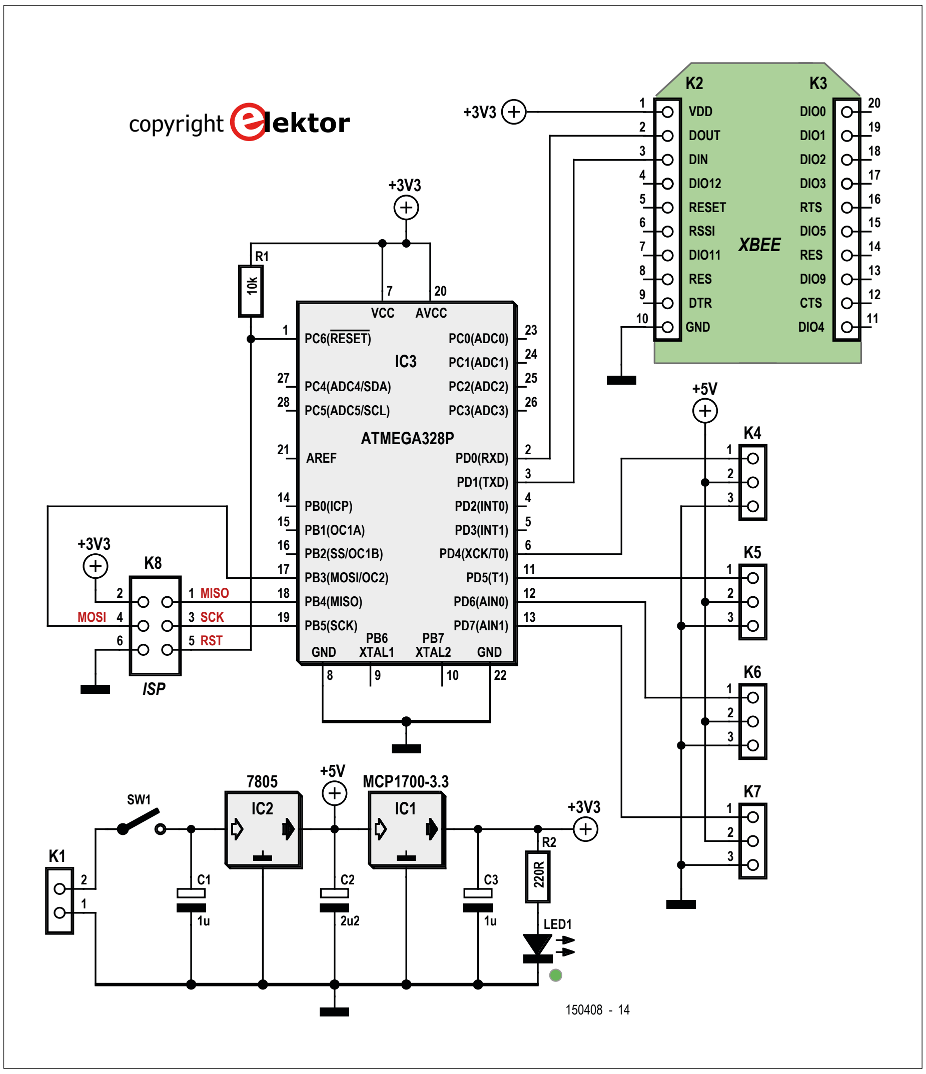


Figure 2. The XBee receiver needs a microcontroller to decode the data received from the transmitter.

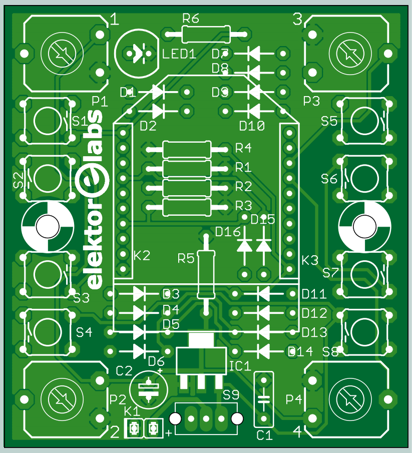
COMPONENT LIST 150408-3 XBee TRANSMITTER

Resistors
P1-P4 = 10kΩ trimmer
R1-4 = 10kΩ
R5 = 120Ω
R6 = 220Ω

Capacitors
C1 = 100nF
C2 = 10µF, 6.3V, 2.5mm pitch

Semiconductors
IC1 = LD1117S12TR
D1-16 = 1N4148
LED1 = LED, green, 3 mm

Miscellaneous
K1 = 2-pin pinheader, 0.1" pitch
K2,K3 = 10-way pinheader socket, 2mm pitch
S1-S8 = tactile switch, 6x6 mm
S9 = slide switch (OS102011MS2QN1C)
XBee S2C or XBee-Pro S2C (+ antenna)



PCB 150408-3
No firmware

Programming the microcontrollers

All three microcontrollers used in this article run from their internal 8-MHz internal RC oscillator, and the configuration fuses must be set in the same way for all of them (see Table 3).
To compile the software for these MCUs in the Arduino IDE it is important to select the correct 'Board' (from the 'Tools' menu). Several possibilities exist, we suggest to choose the 'Arduino Pro or Pro Mini' with the 'Processor' option (from the same 'Tools' menu after you selected the board) set to 'ATmega328 (3.3 V, 8 MHz)'.
You can also opt for the 'Atmega328 on a breadboard (8-MHz internal clock)' as described here: www.arduino.cc/en/Tutorial/ArduinoToBreadboard
Successfully compiling the nRF24 software, furthermore, requires that you install the Arduino RF24 library: https://github.com/TMRh20/RF24.

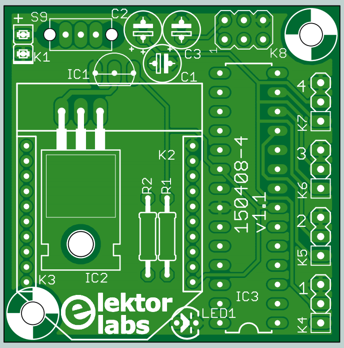
COMPONENT LIST 150408-4 XBee RECEIVER

Resistors
R1 = 10kΩ
R2 = 220Ω

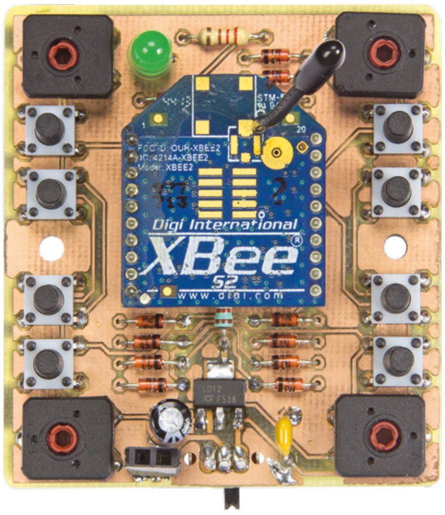
Capacitors
C1,C3 = 1µF, 6.3V, 2.5mm pitch
C2 = 2.2µF, 6.3V, 2.5mm pitch

Semiconductors
IC1 = MCP1700-3302E/TO
IC2 = 7805
IC3 = ATmega328P-PU, programmed (Store # 150408-43)
LED1 = green, 3mm

Miscellaneous
K1 = 2-pin pinheader, 0.1" pitch
K2, K3 = 10-way pinheader socket, 2mm pitch
K4-7 = 3-pin pinheader, 0.1" pitch



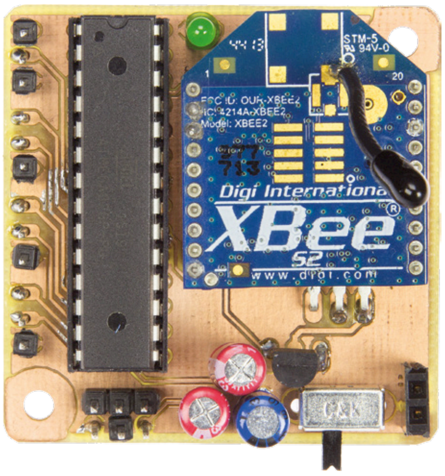
K8 = 6-pin (2x3) pinheader, 0.1" pitch
S9 = slide switch (OS102011MS2QN1C)
XBee S2C or XBee-Pro S2C (+ antenna)
PCB # 150408-4



the received data on its serial port. We assigned the data decoding task to microcontroller IC3, an ATmega328P. Note that when you buy the 'programmable' XBee modules, you can modify the firmware yourself which obviates the need for an external microcontroller. Unfortunately, this approach is only viable for people (i.e. companies) who intend to use the XBee modules on a large scale.
The microcontroller reads the data on its serial port and converts it to servo signals on connectors K4 to K7. IC2 was added to provide a clean 5-V supply for the servos, the rest of the circuit runs off 3.3 V. The software for the receiver was written using the Arduino IDE and is therefore easy to modify (see **inset**).

Connecting the XBee transmitter and receiver

The XBee modules need to be configured before putting them into our desired use. For this you will need a computer and likely a USB-to-serial converter (cable or board, see shopping list), but an XBee breakout board will come in very handy too (build one yourself or see the shopping list for a suitable board). On the



computer you should download and install the free utility XCTU [1]. Be aware that AT-command jockeys can do without XCTU but that's outside the scope of this article.

First we will configure the transmitter module. Connect the module to the PC over a serial link, USB or classic, using a suitable USB-to-serial adapter if needed and an XBee breakout board.

On the computer, launch XCTU. When ready, click the 'Discover radio modules' button. You may have to reset the module before it can be detected. Once the module has been found a configuration screen will open. Click the 'Read' button to read the current module configuration (**Figure 3**).

XBee transmitter configuration

The module must be programmed with the 'ZigBee Router API' function set. To do so, click 'Update', select the 'XB24-ZB' product family together with the 'ZigBee Router API' function set, and firmware version '23A7'. When done, click 'Update', see **Figure 4**.

Now the XBee module can be reconfigured. In the 'Networking' section the 'PAN ID' must be entered. Enter a hexadecimal number between 0 and 0xFFFF FFFF FFFF FFFF (64 bits). This ID is to avoid interference between networks, so better enter something more 'difficult' than 1234. Write it down, because you will also need it for the receiver module. Set 'Channel Verification' to 'Enabled [1]'.

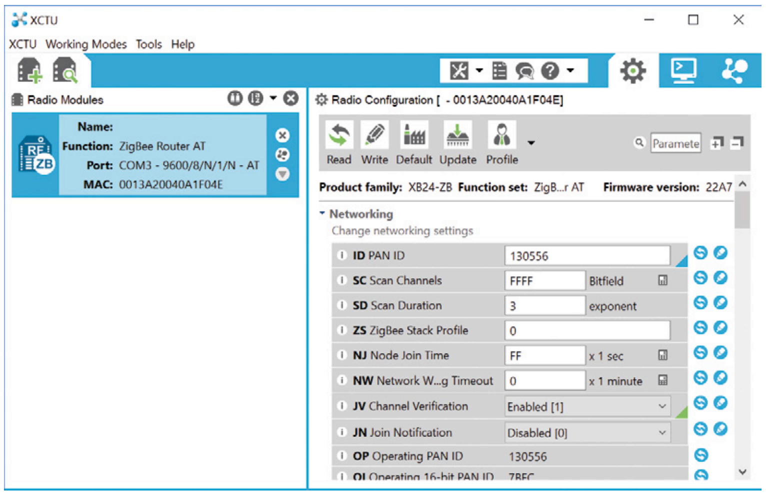
Scroll down to the 'Addressing' section where you have to enter the MAC address of the receiver module. This address is printed on the module, but you can also find it with XCTU. The first (or higher) 32 bits of the 64-bit address go into the 'Destination Address High' box; the last (or lower) 32 bits go into the 'Destination Address Low' box.

The last things to configure are the transmitter module's I/O pins, the so-called 'I/O settings' (scroll almost all the way down to see them). **Table 2** and **Figure 5** show the values that work we used. Set the 'I/O Sampling' value to 64 (below the 'IO settings').

When done, click 'Write'.

XBee receiver configuration

In your XBee module programming setup, replace the transmitter module by the receiver module. Connect the


XCTU
XCTU Working Modes Tools Help
Radio Modules
Read Write Default Update Profile
Parameter

Radio Configuration [- 0013A20040A1F04E]

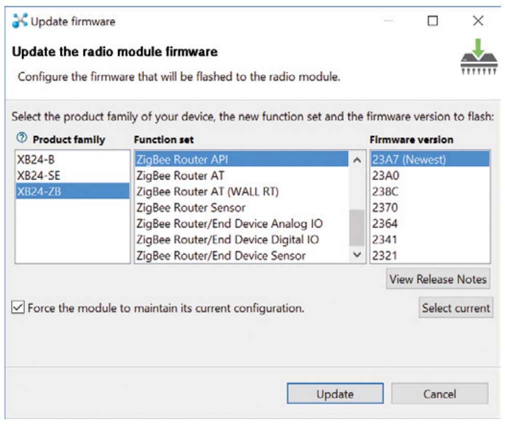
Name:
Function: ZigBee Router AT
Port: COM3 - 9600/8/N/1/N - AT
MAC: 0013A20040A1F04E

Product family: XB24-ZB Function set: ZigB...r AT Firmware version: 22A7

Networking
Change networking settings

ID PAN ID	130556	
SC Scan Channels	FFFF	Bitfield
SD Scan Duration	3	exponent
ZS ZigBee Stack Profile	0	
NJ Node Join Time	FF	x 1 sec
NW Network W...g Timeout	0	x 1 minute
JV Channel Verification	Enabled [1]	
JN Join Notification	Disabled [0]	
OP Operating PAN ID	130556	
OI Operating 16-bit PAN ID	7BEC	

Figure 3. XCTU found an XBee module on COM3.



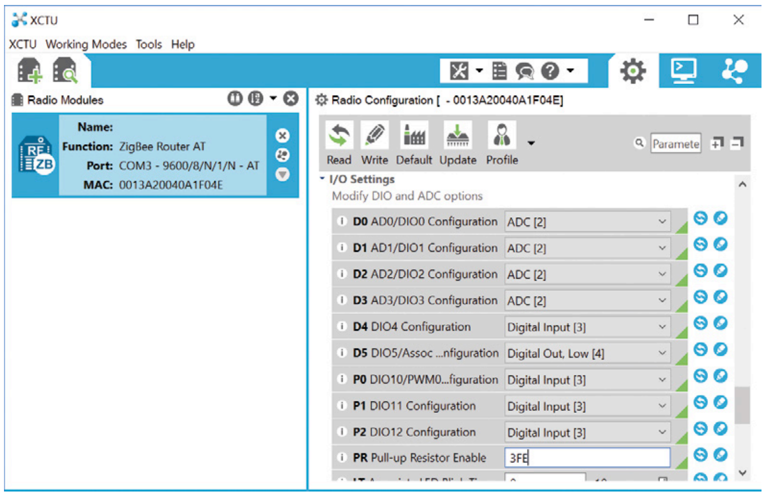
Update the radio module firmware
Configure the firmware that will be flashed to the radio module.

Select the product family of your device, the new function set and the firmware version to flash:

Product family	Function set	Firmware version
XB24-B	ZigBee Router API	23A7 (Newest)
XB24-SE	ZigBee Router AT	23A0
XB24-ZB	ZigBee Router AT (WALL RT)	238C
	ZigBee Router Sensor	2370
	ZigBee Router/End Device Analog IO	2364
	ZigBee Router/End Device Digital IO	2341
	ZigBee Router/End Device Sensor	2321

☑ Force the module to maintain its current configuration.

Figure 4. The XBee transmitter needs the 'ZigBee router API' function set.


XCTU
XCTU Working Modes Tools Help
Radio Modules
Read Write Default Update Profile
Parameter

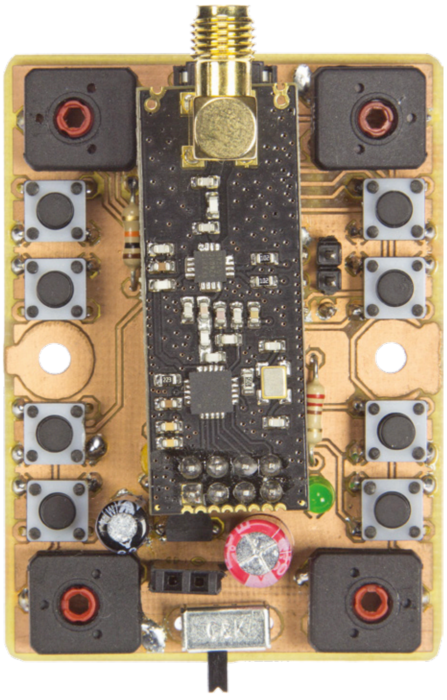
Radio Configuration [- 0013A20040A1F04E]

Name:
Function: ZigBee Router AT
Port: COM3 - 9600/8/N/1/N - AT
MAC: 0013A20040A1F04E

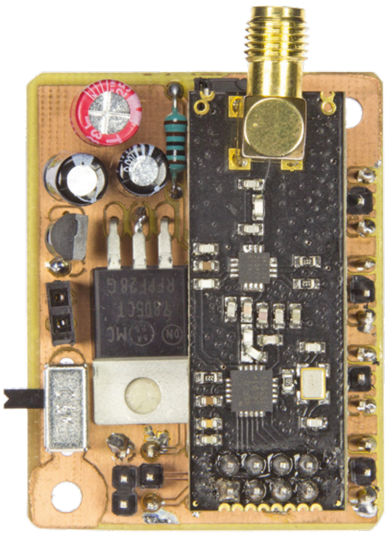
I/O Settings
Modify DIO and ADC options

D0 AD0/DIO0 Configuration	ADC [2]
D1 AD1/DIO1 Configuration	ADC [2]
D2 AD2/DIO2 Configuration	ADC [2]
D3 AD3/DIO3 Configuration	ADC [2]
D4 DIO4 Configuration	Digital Input [3]
D5 DIO5/Assoc ...nfiguration	Digital Out, Low [4]
P0 DIO10/PWM0...figuration	Digital Input [3]
P1 DIO11 Configuration	Digital Input [3]
P2 DIO12 Configuration	Digital Input [3]
PR Pull-up Resistor Enable	3FE

Figure 5. Configuration of the XBee transmitter IO pins.



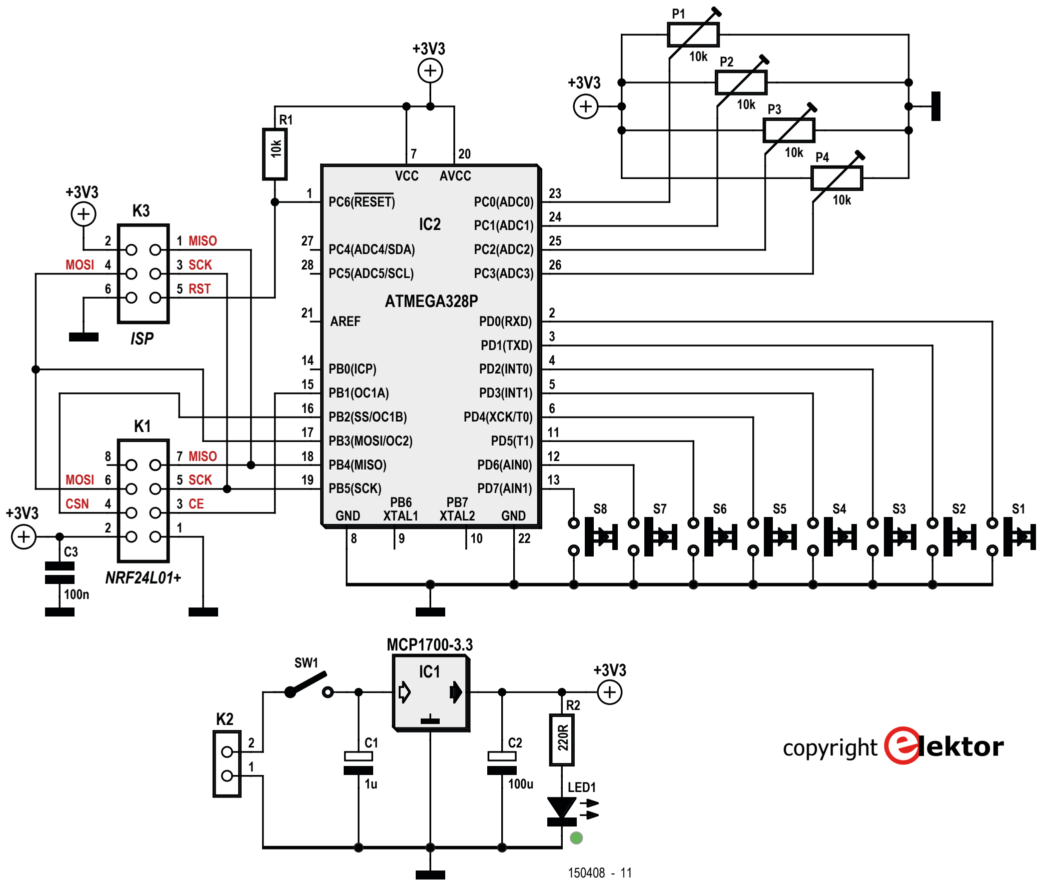
setup to the computer, and click the XCTU ‘Discover radio modules’ button followed by a click on ‘Read’ once it has been discovered.
The receiver module should be loaded with the ‘ZigBee Coordinator API’ function set. Click ‘Update’, select as product family the ‘XB24-ZB’, choose the ‘ZigBee Coordinator API’ function set with firmware version ‘21A7’ and click the ‘Update’ button.
In the ‘Networking’ section enter the same PAN ID as used for the transmitter. In the ‘Addressing’ section a bit further down you should now enter the address of the transmitter module in the same way as you did for the receiver module’s address during the transmitter configuration.
When done click ‘Write’.
Before the receiver will work you should, of course, also program its microcontroller — you can download the firmware from [2]. An AVR ISP programmer is needed for this, connected to K8. Do not forget to set the MCU’s configuration fuses — they can be found in **Table 3**.
Stick the XBee modules on the boards without mixing them up. Also connect the servos to the receiver board. Now power the transmitter and receiver in arbitrary order. Once the connection between the two boards has been established (this



may take a few seconds) you should be able to control the servos by turning the trimpots. If you connect the transmitter to a serial terminal, feedback messages may be displayed.

And now with nRF24 modules...

An RC based on XBee modules is certainly interesting as the XBee hardware allows for all sorts of network config-



copyright elektor

Figure 6. The nRF24 transmitter uses an SPI bus for communication between the MCU and the radio module.

urations, but is a bit on the dear side. A cheaper solution is to use low-cost nRF24L01 radio modules available all over the Internet for little money in place of the XBee modules. With a good antenna, communication distances similar to an XBee-Pro link can be achieved. Even though an external microcontroller is needed to make these modules work, the total cost will be less than for the XBee approach.

Table 1. Adjusting servo offsets.

	'1'	'0'	'TRIM_DOWN'	'TRIM_UP'	Servo offset
S1	0	0	0	1	Channel 1 up
S2	0	0	1	0	Channel 1 down
S3	0	1	0	1	Channel 2 up
S4	0	1	1	0	Channel 2 down
S5	1	0	0	1	Channel 3 up
S6	1	0	1	0	Channel 3 down
S7	1	1	0	1	Channel 4 up
S8	1	1	1	0	Channel 4 down

The nRF24 transmitter

Figure 6 shows the schematic of the nRF24-based transmitter. It shows the same trimpots and pushbuttons as the XBee transmitter (cf. Figure 1), but the XBee module got replaced by an ATmega328 microcontroller together with K1 that connects to the nRF24 radio module. Note that the communication between the MCU and the radio module is over an SPI bus and not an asynchronous serial port.

Because the MCU has many I/O ports the diodes to encode the pushbuttons are not needed. Also the 1.2-V power supply is no longer needed as everything can be powered from one 3.3-V rail.

The nRF24 receiver

The schematic of the nRF24 receiver (**Figure 7**; and our assembled prototype in **Figure 8**) is almost identical to the XBee receiver (cf. Figure 2), except that the XBee module has been replaced by K1 that in turn connects to the nRF24 radio module. As for the nRF24 transmitter, communication between the radio module and the MCU is over SPI instead of through asynchronous serial ports.

Connect the nRF24 transmitter and receiver

This step is pretty simple as it has been taken care of in the firmware of the transmitter and the receiver. All you have to do is program the microcontrollers with the right software. See the **inset** for some important details.

To avoid interference between multiple nRF24 RC systems you can change the network addresses used by a system. At the top of both the transmitter and the receiver sketch you will find a line like this:

```
uint64_t address[] = {0xFFFFFFFF,
   0xFFFFFFFE};
```

In one of the two files replace the values above by two other 64-bit values (whatever you like) while making sure that the two values are not the same, and then copy the modified line to the other file.

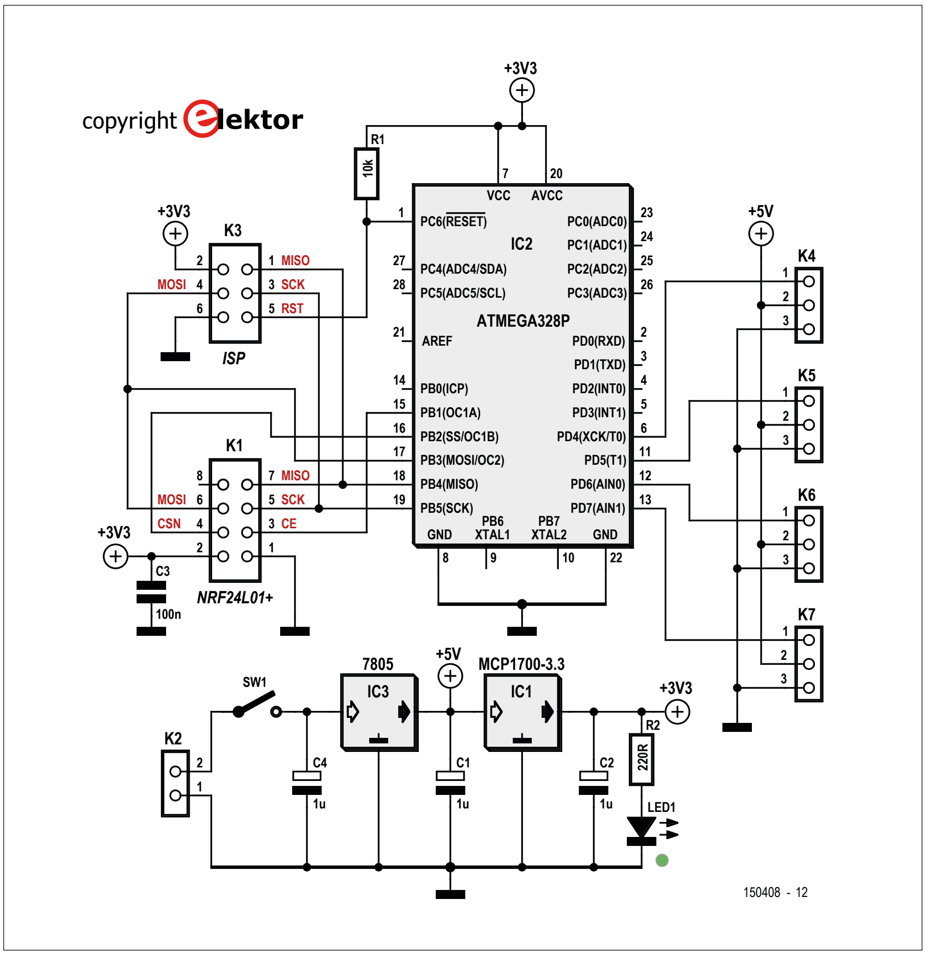


Figure 7. The nRF24 receiver is almost identical to the XBee receiver.

Table 2. Configure the XBee transmitter I/O ports like this.

I/O	Value
D0	ADC [2]
D1	ADC [2]
D2	ADC [2]
D3	ADC [2]
D4	Digital Input [3]
D5	Digital Out, Low [4]
P0	Digital Input [3]
P1	Digital Input [3]
P2	Digital Input [3]
PR	3FE

Table 3. Fuse settings for all the microcontrollers used in this article.

Fuse	Value
Low	0xE2
High	0xDA
Extended	0x05

Recompile, reprogram and you're done. Connect the servos to the receiver board and power the transmitter and receiver in arbitrary order. Once the connection between the two has been established (this may take a few seconds) you should be able to control the servos by turning the trimpots. If you connect the transmitter to a serial terminal feedback messages may be displayed.

Sounding off

Presented were two basic architectures for remote control systems with two-way data communication: XBee and nRF24. Which one is best for you depends on your application and budget. The software implements only basic servo control, it will have to be extended and modified for more elaborate systems. Luckily, this can be done easily as the programs are open source and suitable for the Arduino IDE. ◄

(150408)

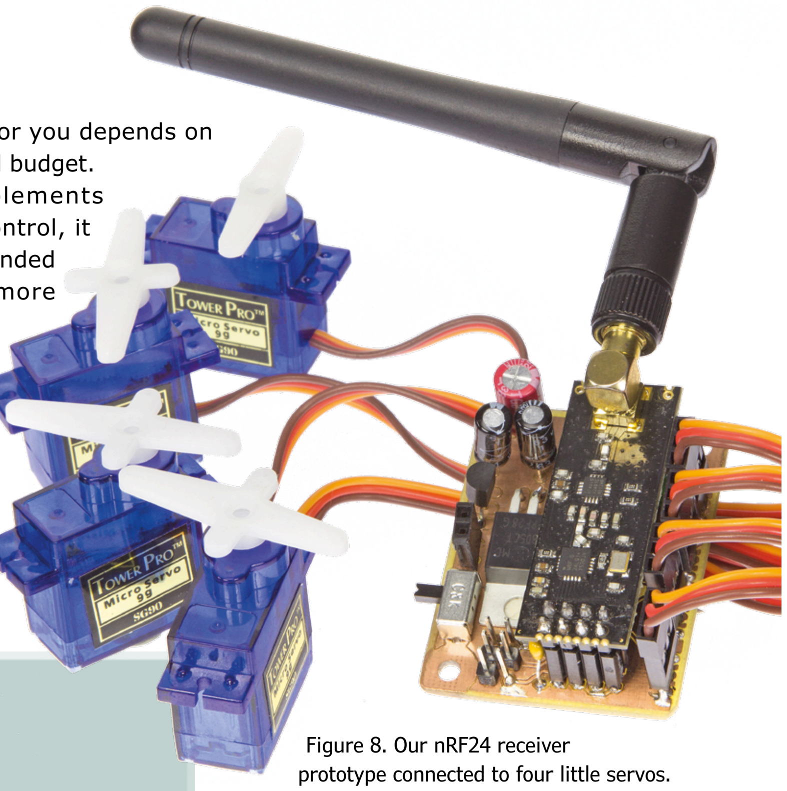


Figure 8. Our nRF24 receiver prototype connected to four little servos.

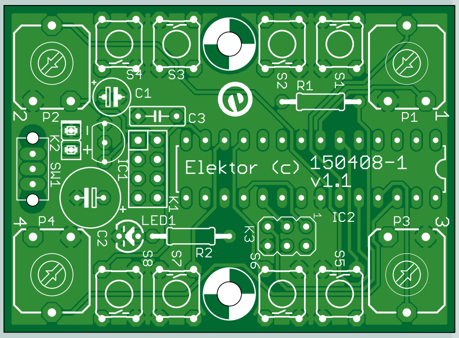
COMPONENT LIST 150408-1 nRF24 TRANSMITTER

Resistors
P1-P4 = 10kΩ trimpot
R1 = 10kΩ
R2 = 220Ω

Capacitors
C1 = 1µF, 6.3V, 2.5mm pitch
C2 = 100µF, 6.3V, 3.5mm pitch
C3 = 100nF

Semiconductors
IC1 = MCP1700-3302E/TO
IC2 = ATmega328P-PU, programmed (Store # 150408-41)
LED1 = green, 3mm

Miscellaneous
K1 = 8-way (2x4) pinheader socket, 0.1" pitch
K2 = 2-pin pinheader, 0.1" pitch
K3 = 6-pin (2x3) pinheader, 0.1" pitch
S1-S8 = tactile switch, 6x6 mm
SW1 = slide switch (OS102011MS2QN1C)
nRF24L01+PA+LNA (+ antenna)
PCB # 150408-1



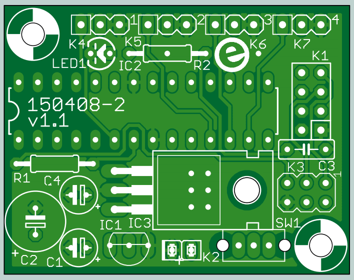
COMPONENT LIST 150408-2 nRF24 RECEIVER

Resistors
R1 = 10kΩ
R2 = 220Ω

Capacitors
C1,C4 = 1µF, 6.3V, 2.5mm pitch
C2 = 100µF, 6.3V, 3.5mm pitch
C3 = 100nF

Semiconductors
IC1 = MCP1700-3302E/TO
IC2 = ATmega328P-PU, programmed (Store # 150408-42)
IC3 = 7805
LED1 = green, 3mm

Miscellaneous
K1 = 8-way (2x4) pinheader socket, 0.1" pitch
K2 = 2-pin pinheader, 0.1" pitch
K3 = 6-pin (2x3) pinheader, 0.1" pitch
K4-K7 = 3-pin pinheader, 0.1" pitch
SW1 = slide switch (OS102011MS2QN1C)
NRF24L01+PA+LNA (+ antenna)
PCB # 150408-2



Web Links

[1] www.digi.com/products/xbee-rf-solutions/xctu-software/xctu

[2] www.elektormagazine.com/150408

FROM THE STORE

nRF24 version

→150408-1
Transmitter PCB

→150408-2
Receiver PCB

→150408-41
Programmed transmitter microcontroller

→150408-42
Programmed receiver microcontroller

XBee version

→150408-3
Transmitter PCB

→150408-4
Receiver PCB

→150408-43
Programmed receiver microcontroller

Tools

→140374-91
T-Board Wireless

→150387-1
Hi-speed USB UART

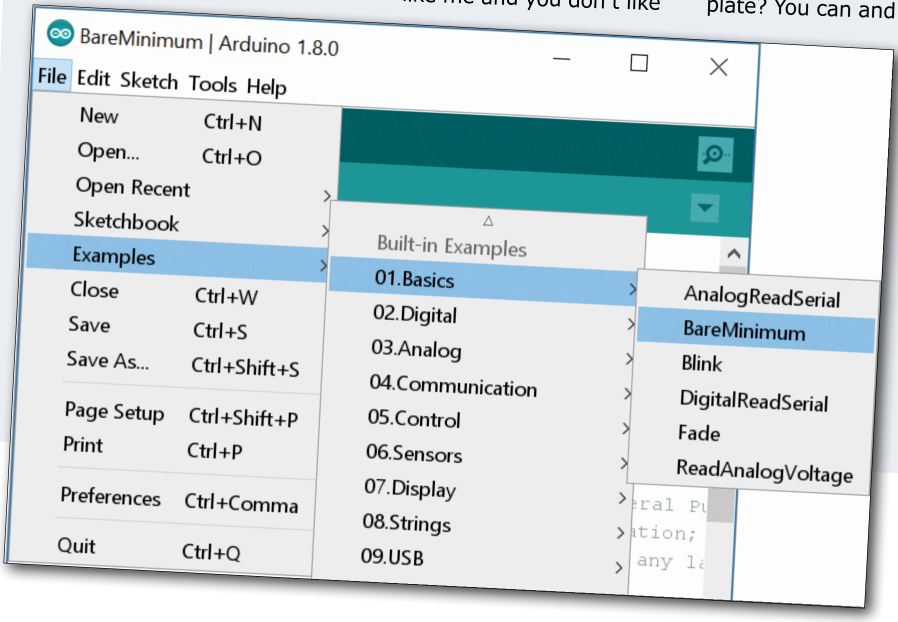
HomeLab Helicopter

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

Undocumented Arduino: new sketch template

When you open the Arduino IDE or when you create a new sketch from the File → New menu the editor already contains empty skeletons for the functions `setup` and `loop`. If you are like me and you don't like the way these skeletons are written you have to edit them every time, which is almost as time consuming as writing the skeletons from scratch. If only you could change this template? You can and here's how. Open the file `<arduino>\examples\01.Basics\BareMinimum\BareMinimum.ino` in Notepad or some other basic text editor, then modify and save it. Next time you create a new sketch it will contain your template. You cannot do this in the IDE itself because it will try to protect the example from being overwritten. A custom template is a great way to automatically add the lines of code (like `Serial.begin(115200);`) or license text you use in every sketch.

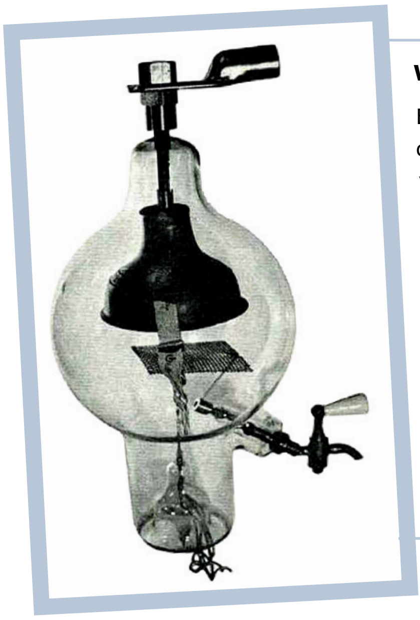
(160354-a)



Must-have homelab tool

Homelabbers love gadgets and handy tools (duh, is the atomic number of silicon 14?) and so we always keep an eye open for good candidates to add to the already impressive piles on their desks. This time we would like to present a solar-powered tool that should please the maker that leaves his workspace every once in a while (yes, they do exist!), maybe to go hiking in the wild where they might get attacked by mosquitos and maybe even get lost. If this happens the compass will help them find their way back safely without being bitten by annoying insects. Its handy carabiner shape attaches to any belt loop, making it easy to carry wherever you go. (160354-d)

Digisonic 12D8 solar-powered mosquito repeller with compass, available all over the Internet.



The Umac 606 Phantasatron.

Weird components

Elsewhere in this edition you can read about the Phantastron, a special multivibrator circuit with very precise timing invented during the Second World War. The Phantastron is a very clever circuit indeed, but I am sure that if its creator Alan Dower Blumlein had known about the Umac 606 Phantasatron the circuit had been even better. The Umac 606, commercialized at about the same time, was an infernal anode, helical beamed phantasatron with urinated tungsten filament obtained through a special process with triple-distilled, single-isotope uranium of which all neutrons had been removed. The grid was constructed of the rare metal senileium, chosen for its total lack of emission. A double-pumped vacuum permitted a clear view of the non-emitting triple-processed plunger-type plate. Because of its unique self-flushing construction, this tube remained usable throughout its useful life. For more details please refer to the datasheet.

www.tubecollectors.org/archives/606.pdf

(160354-c)

KiCad EDA 3D library

Over the past decades the open-source software community has produced many extremely good products, some of which are totally free, even for commercial applications. Created and maintained by thousands of enthusiasts, in teams as well as by individuals who hardly ever meet in real life, one can only admire the energy and effort that these people put in these projects without expecting anything in return. Linux is probably the first example that comes to mind, but there are many others. KiCad EDA, pronounce as ‘Keekad’, a cross-platform electronics design automation suite, is another. According to Wikipedia the project was started in 1992 by Jean-Pierre Charras. My first experiences with KiCad date back to 2003 or so. At the time it was usable, but not adapted to my day job. However, KiCad has kept evolving and today, 25 years old, it has become a professional-grade schematics and PCB design tool with 3D viewing capabilities.

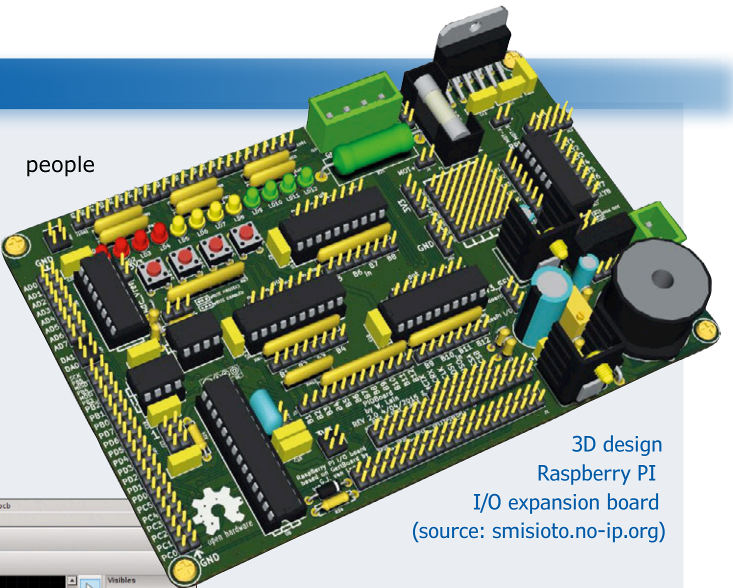
As happens with such successful community projects, many people contribute. In the case of KiCad contributions are mainly in the form of component libraries and tutorials, but also functional extensions are added like the push-and-shove interactive router developed by the very serious Hardware and Timing team at the CERN international nuclear research labs. My own contributions to KiCad are inexistent — only this short piece really- which I use to spread The Word.

Recently my attention was drawn to an Italian website maintained by one Walter from Padua, offering a very large 3D library of components for KiCad. Besides the libraries, the website also explains some techniques for designing nice 3D component models so be sure to have a look if you have to design your own.

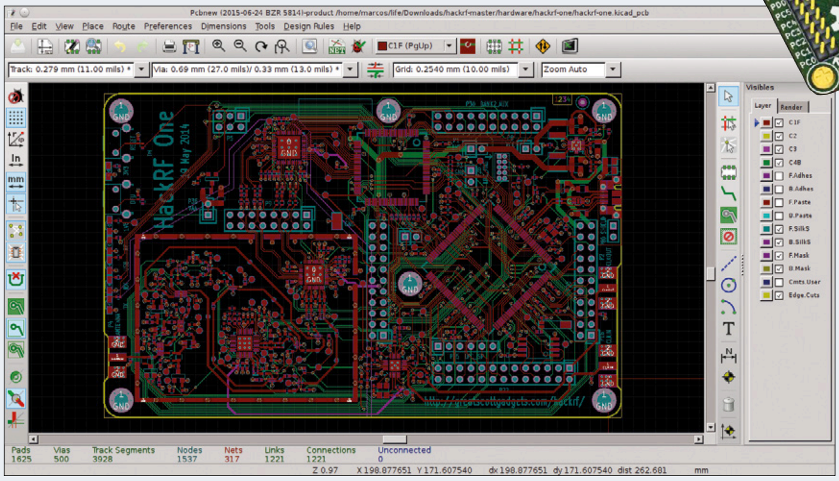
http://smisioto.no-ip.org/elettronica/kicad/kicad-en.htm

http://kicad-pcb.org/

(160354-b)



3D design
Raspberry PI
I/O expansion board
(source: smisioto.no-ip.org)


File Edit View Place Route Preferences Dimensions Tools Design Rules Help
HackRF One
Pads 1625 Vias 500 Track Segments 3928 Nodes 1537 Nets 317 Links 1221 Connections 1221 Unconnected 0

KiCad’s PCB editor PcbNew (source: kicad-pcb.org)

Can you trust your personal assistant?

I think it started a few years back with Apple's Siri, the "intelligent" personal assistant and knowledge navigator built in for instance iPhones. Of course, the competition quickly followed the example and today we also have Microsoft's Cortana, Google Assistant, and Samsung Viv. They all have their quirks and strong points, but we can expect them to improve a lot in the (near) future. Surprisingly, the assistant that is currently having the best chances of beating them all is Amazon's Alexa. Implemented by the highly successful smart, voice-enabled speaker 'Echo' and its relatives 'Echo Dot' and 'Tap', Alexa was present in more than 30 new products shown at the 2017 Consumer Electronics Show (CES) held every year at Las Vegas. Voice-enabled here does not mean that it can also make vocal sounds audible, but that it is controlled by voice. You tell it to play Elvis, and it will play Elvis.

The success of Alexa is due to its openness. Where Siri and Cortana were initially closed, Alexa has been an open platform right from the start, actively inviting developers to add "skills" to it. A skill is a new function, like ordering a pizza hands-free or changing the color of an LED simply by saying so. Google Assistant is also an open platform, allowing third party "actions". Forced by the success of Alexa, Microsoft and Apple have recently added developer access to Cortana and Siri too.

Contrary to Siri and Cortana that use the processing power of their host to do the job, Alexa uses Amazon's cloud services for the heavy lifting, making it a light-weight voice-control system for all sorts of applications. Cloud-based implies an Internet connection, without one it will not work. A system using Alexa starts listening when it hears the wake word 'Alexa' (can be changed by the user). It then starts capturing audio and streams the data over the Internet to the server where it is analyzed and interpreted. It is also stored there for later reference, to improve its voice recognition capabilities, and this is the point where privacy concerns pop up. Indeed, a device capable of streaming private conversations to the cloud where they are stored is a potential privacy threat. Authorities, or hackers for that matter, might want to tap into such an interesting database for more or less justifiable reasons.

STORING BULK METADATA ISN'T REALLY DOMESTIC SPYING...
WE LEAVE THAT KIND OF DATA COLLECTION TO THE EXPERTS...
... AT GOOGLE, AMAZON, FACEBOOK, TWITTER, INSTAGRAM, APPLE, MICROSOFT, YAHOO...
NSA
NSA
NSA
ROGERS ©2015 PITTSBURGH POST-GAZETTE

ROGERS © 2015 Pittsburgh Post-Gazette. Reprinted by permission of ANDREWS MCMEEL SYNDICATION. All rights reserved.

According to Amazon, requests to access this data have always been refused up to now, but nobody knows what may happen in the future. Users can delete their recordings, but doing so will cripple the intelligent assistant, so few people will probably do this. Are you ready to give up your privacy so you can order a pizza without dialing a number?

(160354-e)

25,000,000,000 microcontrollers were shipped in 2016

Onyx Connect, the first company ever to manufacture smartphones and tablets in Africa

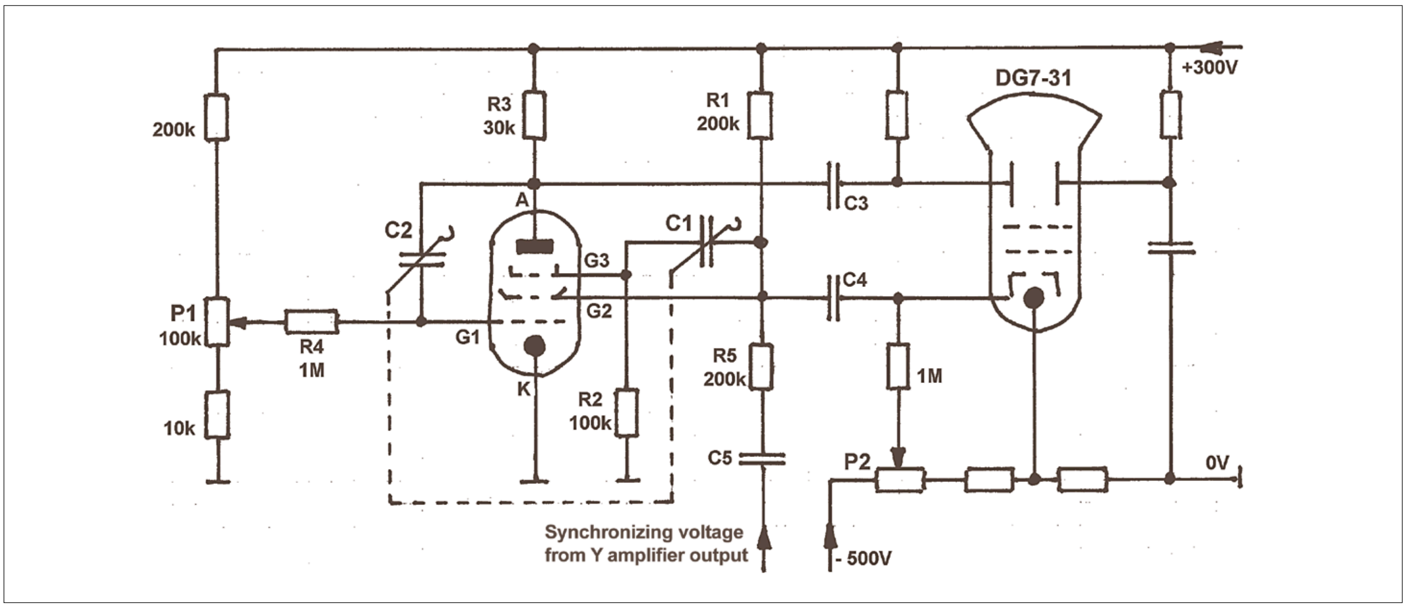
Want to participate? Please send your comments, suggestions, tips and tricks to labs@elektor.com

The Phantastron

Peculiar Parts, the series

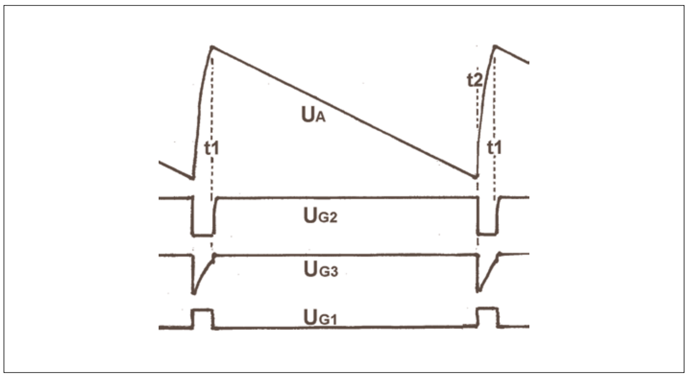
By **Lutz Bergmann** (Germany)

Despite its extreme linearity and accuracy, this remarkable circuit design is all but forgotten today. It deserves greater recognition.



Certain electronic components share a name ending in –tron, such as the ignitron, magnetron, thyratron and klystron. Each of these could be described as peculiar parts, which is why the klystron has already been covered in this series, as was the Dekatron, Trochotron… Unlike the other –trons, the subject of this article is not a physical component but a very peculiar and special kind of circuitry. It was designed in 1940 by a very special personage — Alan Blumlein, the inventor of stereophonic sound and many other advances in telecommunications, sound recording, television and radar [1].

In the phantastron a pentode is driven in a way that is carefully avoided in conventional amplifier circuits. A curious effect is employed in a fantastic manner for generating sawtooth voltages [2, 3]. This special effect relates to the fact that the current in the screen grid of a pentode increases strongly when the plate (anode) voltage becomes smaller than the screen grid voltage. The electrons are attracted less towards the plate and more to the screen grid, because this has the higher voltage. If the screen grid is supplied with current via a resistor, then this moment is recognizable by the fact that the screen grid voltage, which previously was independent of the anode voltage, now begins to fall. This negative change can be transferred to the suppressor grid via capacitor C1. This then acts as an additional control grid and inhibits flow of the last remaining anode current by the now negative control voltage. The anode current is switched off abruptly. The voltage at the anode can rise rapidly once more. With this bistable operation, a so-called Miller Integrator can be expanded into a self-exciting sawtooth oscillator.



Let's start at the end of an off-duration, which we will call t1. The plate voltage is at its maximum, because plate current is zero. Because the coupling capacitor C1 can hold the suppressor grid voltage negative for only a limited time, plate current begins to flow again. The drop in voltage on resistor R3 rises and the plate voltage U_A falls. This change in voltage causes a circulating current at C2 that leads to a negative change in voltage at control grid G1, which moderates the rise in plate current. Equilibrium arises between the current in C2 and R4. Since the current in R4 is constant, the same applies in C2. The plate voltage falls in desirably time-linear fashion. This effect was discovered in 1919 by John Milton Miller [4]. The rate at

which the plate voltage falls can be set infinitely variably using potentiometer P1 in a 1:10 ratio and in steps by selecting the value of capacitor C2. This operation repeats for as long as it is not disturbed until the plate voltage becomes less than the screen grid voltage and the plate current is thereby inhibited. The plate voltage rises once more, although not at arbitrary speed, since C2 must be recharged. The time constant C2 x R3 is influential in this. The voltage on the suppressor grid G3 must remain negative long enough for this. Consequently C1 must be matched to C2. If the value of C2 is altered by means of a selector switch, then the same must happen for C1.
After 1945 this sawtooth oscillator had significant importance for the construction of simple oscilloscopes, as in those days vacuum tubes were still very expensive. With just one single pentode it became possible in a fantastic manner to provide all the signals necessary for the horizontal (x-direction) deflection of a cathode ray tube (CRT). The sawtooth voltage developed 200 V peak-to-peak without additional components for the horizontal deflection of a DG7-31 CRT. By the use of capacitor C3 this could be applied to one of the two horizontal deflection plates (asymmetrical triggering for a DG7-31 CRT with 7 cm screen diameter). A negative voltage pulse could be tapped off the screen grid via capacitor C4 for the Wehnelt Cylinder [5] of the CRT, to enable screen blanking during beam flyback.
An oscilloscope as basic as this offered little in the way of operating convenience. The horizontal deflection was fundamentally free-running. The sawtooth oscillator frequency had to be adjusted to the frequency of the signal being examined in order to achieve a static display. For synchronizing the output of the vertical (y-direction) amplifier was linked to the screen grid via an R-C element R5/C5.
With the advance of triggerable horizontal deflection and the fall in prices of vacuum tubes the Phantastron disappeared off the screen, so long ago that hardly anybody remembers it now. At the same time transistors and subsequently integrated circuits as well have driven vacuum tubes into obscurity, whilst sweep generators have improved beyond recognition along with the ease of triggering.
Further information can be found on the Internet, including the solid-state version of this circuit [6] and the bewilderment arising when the circuit had its Secret status declassified [7]. ⏮

(160281)

Web Links

[1] Alan Blumlein: https://en.wikipedia.org/wiki/Alan_Blumlein

[2] Miller transistron: www.r-type.org/articles/art-135.htm

[3] Miller timebase: www.r-type.org/articles/art-136.htm

[4] Miller effect: https://en.wikipedia.org/wiki/Miller_effect

[5] Wehnelt cylinder: https://en.wikipedia.org/wiki/Wehnelt_cylinder

[6] Radar Basic website: http://www.radartutorial.eu/17.bauteile/bt52.en.html

[7] Security (p. 380): http://americanradiohistory.com/Archive-Wireless-World/40s/Wireless-World-1946-11.pdf

Advertisement

IoT Design Contest
Curation is Creation
2017.5.1 ~ 8.31 | Total Prize of $15,000
WIZnet OSHW MAKER SPACE MUSEUM
wiznetmuseum.com
For Makers, By Makers
IoT UCC Platform
cybermakerspace.com
WIZwiki_W7500
ARM mbed enabled
WiFi Shield
WizFi310
WIZnet IOP 4 IoT
IOP(Internet Offload Processor), TCP/IP hardware implementation, is fast, safe and reliable to perform the data transfer by fully offloading processing of the entire TCP/IP stack without host CPU intervention.
WIZnet.io

HOMELAB PROJECT

IoT Gateway and Wireless Nodes (2)

Part 2: the software

The author has looked in vain for a suitable commercial home automation system. Since he couldn't find anything to his liking, he decided to develop his own system. Various 'end nodes' (sensors and actuators) communicate via an 868 MHz radio link with a central gateway. The gateway uses MQTT to send the measurement values to an OpenHAB server, which processes them into a form suitable for display [1]. The construction of the hardware was extensively covered in the previous article [2]. This article concerns itself with the software that makes everything tick.

By **Hennie Spaninks** (The Netherlands)

As we mentioned in the previous part, we used the software on the following platforms. The Arduino sketches were written by the author himself, who naturally made use of the various Open-Source Libraries available.

End node
The Arduino sketch for the end node: this code controls the radio link, reads the sensors and controls the outputs.

Gateway
The Arduino sketch for the gateway: this code controls the radio link and decodes the messages. An MQTT client is used to exchange messages with the broker.

Home automation server (Raspberry Pi)
The Raspberry Pi is used to run the MQTT broker (Mosquitto) as well as the OpenHAB home automation controller.

A standard Arduino development environment was used to compile the Arduino code, extended with a number of libraries, which are described later on. It's best to use the most powerful version of the Raspberry Pi (RPi 3), although the Raspberry Pi 2 would probably cope as well. We use Raspbian for the OS.

End node

The code for the various end nodes can be found at [3]. The core of the code is the same for all of the end nodes. For this description we'll focus on the DHT end node [4]. This node has a digital input (a push-button) on input 8, a digital output (relay/led) at pin 9 and a humidity/temperature sensor at pin 4. When the push-button has been pressed, the node can carry out one of two local functions:

- Start a timer and set the output high. Reset the output when the timer has finished. The timer period is set via device 7 (see Table 2 in part 1). A value of '0' disables the timer function.
- Toggle the output whenever the push-button is pressed.

This enables you to manually turn the output on and off locally. You have to set device 6 to 'ON' to enable this function.

Completely separately from these two functions, a status message is transmitted via device 40 whenever the level changes at the digital input (pin 8). The values of the humidity/temperature sensor can be read via device 48/49. Regular transmissions are also possible.
The code makes use of three libraries:

- RFM69 library by Felix Ruso [5].
- SPI library to control the SPI bus.
- DHT library to read the DHT11 sensor.

The code begins with a number of configuration parameters (from line 60 onwards):

```
#define NODEID 2        // unique node ID within
   the closed network
#define GATEWAYID 1     // node ID of the Gateway
   is always 1
#define NETWORKID 100   // network ID of the
   network
#define ENCRYPTKEY "xxxxxxxxxxxxxxxx"   // 16-char
   encryption key; same as on Gateway!
//#define DEBUG                         //
   uncomment for debugging
#define VERSION "DHT V2.2"              // this
   value can be queried as device 3
```

The NODEID gives the end node a unique address within the radio network. This ID is also used in the MQTT topic to address the required node. The nodeID of the gateway is always '1'. The radio link from the end node is always made with the gateway. There is therefore never any traffic directly between individual end nodes.
The NETWORKID is used to create closed networks. Only those nodes that are within the same network can then communicate with each other. We're using a closed network in this project, which means that the network ID has to be identical in the gateway and all of the end nodes.
The ENCRYPTKEY is the key used to encrypt the radio traffic. This key consists of 16 characters and has to be identical for all nodes in the network.
The DEBUG parameter is a switch to enable the debugging mode. When this parameter has not been defined (by setting this line as a comment by preceding it with '//'), no debug information will be generated. When the '//' is removed, DEBUG becomes active and status messages will be transmitted via the serial port.

Several other parameters are required for the RFM69 module to function correctly:

```
#define FREQUENCY RF69_868MHZ
#define IS_RFM69HW   // uncomment only for RFM69HW!
#define ACK_TIME 50  // max # of ms to wait for an
   ack
```

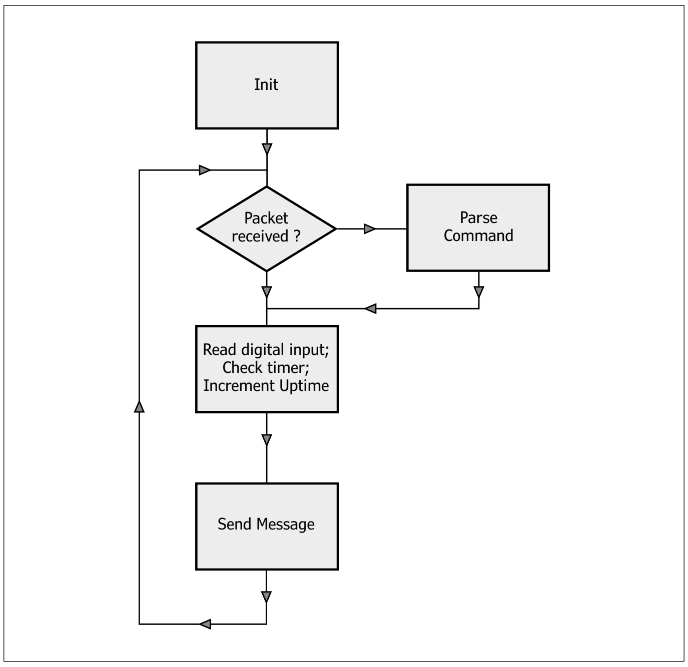


Figure 1. Program flow of the end node.

FREQUENCY is the frequency used, which in Europe is either 433 MHz or 868 MHz. IS_RFM69HW states that the High Power version is used. ACK_TIME is the maximum time that the module waits for an acknowledge after the transmission of a message.

From line 78 onwards are several definitions for the DHT sensor, the pin connections used, and for some variables. The comments in the source code give more details of the function of these variables. Following line 135 is the initialization of the program (setup()), where the output pin is configured and the radio module is initialized.

The program flow is shown in the flowchart in **Figure 1**.
First of all, (in line 167) we check if the RFM69 module has received a radio packet. If so, the function parseCmd is called. A certain send flag is set, depending on the deviceID and the R/W command in the received data block. Next, we check if the push-button status has changed or if the timer period has completed. After that, the uptime counter is incremented, if required. When the interval for periodic transmissions has expired, the send flags of these parameters are set.
And finally in line 235, the function sendMsg is called. In here, a number of radio packets are prepared depending on which send flags are active. The send flag is then reset and the function transmits the packet by calling txRadio().
If the temperature or humidity is required (device 48 or 49), the value is obtained from the DHT module in line 478 or 485.

Gateway

The gateway has the following functions:
- Receive an MQTT message, decode it, create a radio packet and transmit it.
- Receive a radio message, decode it, create an MQTT message and publish it.
- Receive, decode and answer requests to the gateway itself, such as uptime, version information, etc.

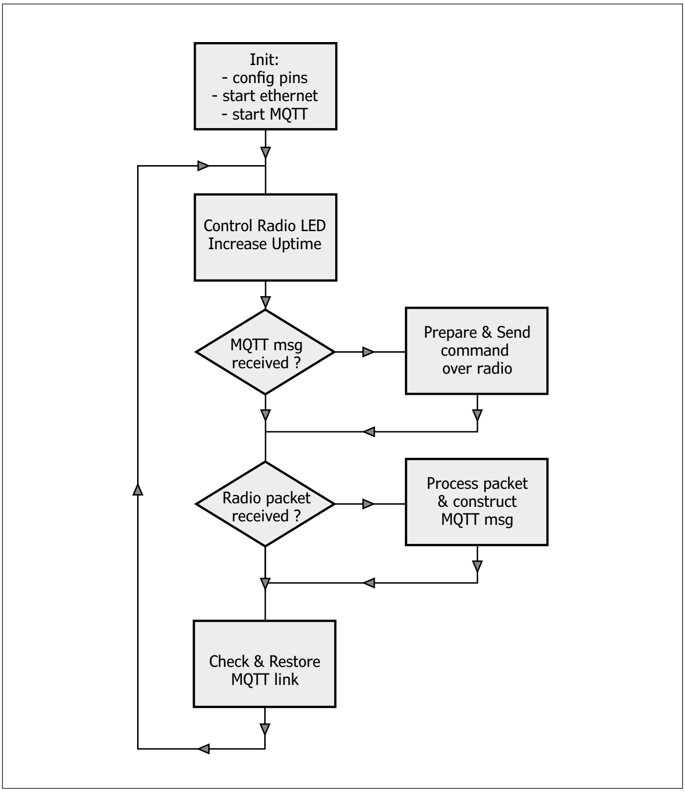


Figure 2. Program flow of the gateway.

The gateway has a fixed Ethernet connection and obtains its network information via DHCP. The gateway has two leds. The radio led is lit when the radio connection is active. The MQTT led is lit when there is a connection to Mosquitto.
Since the gateway program takes up a lot of memory, the debug function has been split into two parts: one part to debug the radio connection and another part to debug the connection to the MQTT broker. Only one of these can be active, otherwise the program won't compile.
The description of the program is for the latest version (2.4 as we go to press), which can be found at [6].

The gateway sketch makes use of the following libraries:

- *RFM69*
- *SPI*
- *Ethernet*
- *PubSubClient* [7]

There are several differences in the gateway sketch, compared to the standard settings:

- The SPI bus is used by two devices in the gateway. Both devices use '10' for the default device address. We selected devicecode 8 for the RFM module and 10 for the Ethernet module. This configuration is selected in line 80 of the gateway code.
- The RFM module uses interrupts to indicate when a radio packet has been received. As long as the Ethernet driver (W5100.h) is active on the SPI bus, these interrupts won't be processed correctly, which causes the gateway to hang. We make use of a solution suggested by Martin Harazinov and make some changes to the file w5100.h to prevent this from happening (see [8]). In the file w5100.h (which can be found in *../Documents/Arduino/libraries/Ethernet/src/utility*) the lines from line 342 have `cli()` and `sei()` statements added to them.

```
#else
inline static void initSS()   { DDRB |= _BV(2); };
inline static void setSS()    { cli(); PORTB &= ~_
   BV(2); };
inline static void resetSS()  { PORTB |= _BV(2);
   sei(); };
#endif
```

This stops any interrupts from occurring while the Ethernet traffic goes over the SPI bus.

The code in the gateway sketch begins with a declaration section that looks very similar to the one for the end node. All the parameters for the radio link are set here. The following parameters for the Ethernet are also set here:

- `Ip`, for the IP address of the gateway. This is a fallback, since the gateway uses DHCP and therefore gets a dynamically allocated address.
- `Mac`, for the MAC address of the Ethernet connection. Note that this must be unique within an IP subnet, so be careful when you have other Arduinos in the same network.
- `Mqtt_server`, the address of the Mosquitto server (we'll look at this later).

The program flow of the gateway is shown in the flowchart in **Figure 2**. The initialization starts at line 135. The output pins for both leds are configured, the Ethernet connection is started and then the connection to the MQTT broker is started. The MQTT client is initialized in line 129. A subscription is requested that's defined in the function `mqtt_subs` (line 382). The topic of the subscription is *home/rfm_gw/sb/#* (line 121), which causes all SouthBound messages to be received. When MQTT messages are received, the variable `mqttToSend` is set to TRUE. The main program starts with a test to see if the radio led has to be turned off, and the uptime of the gateway is incremented.
In line 223 the variable `mqttToSend` is used to check if there are any messages that need to be sent over the radio network. This is done by the function `sendMsg` (line 248). If necessary, the last message is retransmitted every half second until the reception of the packet has been acknowledged by the end node. This is repeated a maximum of five times.

A check is then made in line 225 to see if a radio packet has been received from an end node. If that's the case, the function `processPacket` (line 288) is called. The MQTT topic from the gateway is always NorthBound and is made up from the nodeID and the deviceID and then stored in the string `buff_topic` in line 303. Depending on the deviceID, the type of data in the radio packet is determined (integer, float, status or string), as is the required format of the MQTT message. The string `buff_mess` will eventually contain the MQTT message. Finally, a check is made in line 227 to see if the MQTT link is still active. If this isn't the case, the link is restarted every two seconds in line 230. The MQTT led is turned on in line 235 if the connection has been made.

OpenHAB

The architecture of OpenHAB is shown in **Figure 3**. Openhab has been designed around an event bus. This bus is used to exchange events between the input and output interfaces and the rest of OpenHAB. This configuration permits the use of several IO interfaces ('bindings') at the same time. You can make specific bindings for various protocols. At this point in time bindings are available for smart meters, Sonos, Philips Hue lamps and also for the Nest thermostat and Tesla electric vehicles, amongst others. The page at [9] has a complete overview of the available bindings.

Installation

For the installation we'll use a Raspberry Pi 3 with a 16 GB SD card. We use version 2 of OpenHAB. The installation of OpenHAB is easiest if you use OpenHABian. This is an image with a minimal linux system in combination with installation scripts. This image can be downloaded from [10]. A suitable tool (WinDiskImager [11], for example) can be used to write this image to an SD card. This is put in the Raspberry Pi, and the installation will start immediately after booting. Be aware that this will take quite some time.

As part of the installation process *Samba* is also installed. This makes the configuration directories of OpenHAB accessible via the network, so you can make simple configuration changes. Once the installation is complete, we can install Mosquitto using an installation tool:

```
pi@openHABianPi: sudo openhabian-config
```

If all is well, you should see the window as shown in **Figure 4**. Option 22 will install Mosquitto. We must ensure that the Raspberry Pi always has the same IP address. This is done by modifying the file */etc/network/interfaces*. You should set `address` and `gateway` to the required values for your configuration, for example:

```
auto eth0
    iface eth0 inet static
        address 192.168.2.7
        netmask 255.255.255.0
        gateway 192.168.2.254
```

The new settings will become active after a reboot.

Configuration

To get OpenHAB to work, we need to configure the MQTT binding. The file that needs to be modified is on the RPi at */etc/openhab2/services/mqtt.cfg*. The broker URL is the only parameter that needs to be changed, as shown here:

```
# Define your MQTT broker connections here for use
   in the MQTT Binding or MQTT
# Persistence bundles. Replace <broker> with an ID
   you choose.
```

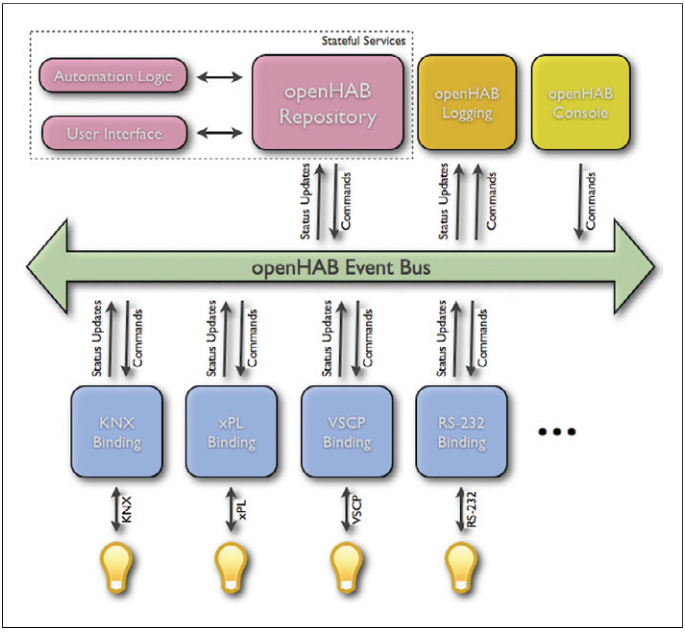
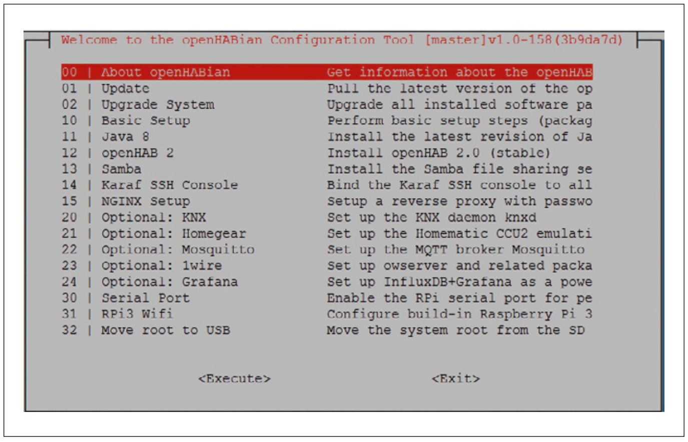


Figure 3. OpenHAB architecture (source: OpenHAB.org).



```
Welcome to the openHABian Configuration Tool [master]v1.0-158(3b9da7d)

00 | About openHABian              Get information about the openHAB
01 | Update                       Pull the latest version of the op
02 | Upgrade System               Upgrade all installed software pa
10 | Basic Setup                  Perform basic setup steps (packag
11 | Java 8                       Install the latest revision of Ja
12 | openHAB 2                    Install openHAB 2.0 (stable)
13 | Samba                        Install the Samba file sharing se
14 | Karaf SSH Console            Bind the Karaf SSH console to all
15 | NGINX Setup                  Setup a reverse proxy with passwo
20 | Optional: KNX                Set up the KNX daemon knxd
21 | Optional: Homegear           Set up the Homematic CCU2 emulati
22 | Optional: Mosquitto          Set up the MQTT broker Mosquitto
23 | Optional: 1wire              Set up owserver and related packa
24 | Optional: Grafana            Set up InfluxDB+Grafana as a powe
30 | Serial Port                  Enable the RPi serial port for pe
31 | RPi3 Wifi                    Configure build-in Raspberry Pi 3
32 | Move root to USB             Move the system root from the SD

                 <Execute>                    <Exit>
```

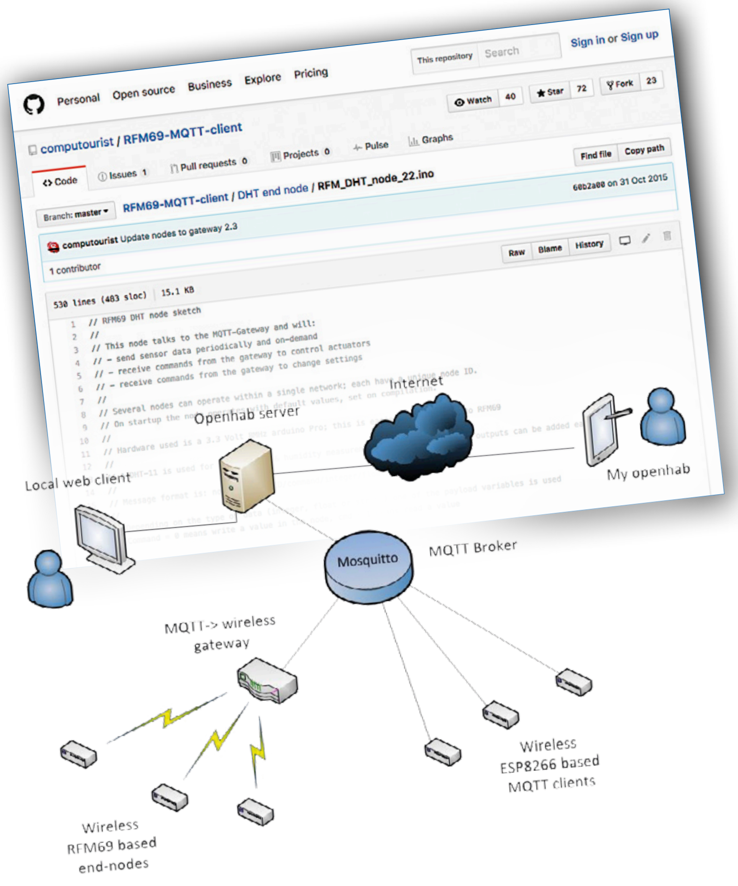
Figure 4. OpenHABian configuration.

```
# URL to the MQTT broker, e.g. tcp://localhost:1883
   or ssl://localhost:8883
mosquitto.url=tcp://localhost:1883
```

This way we point at localhost for the Mosquitto server. Note that 'mosquitto' is the name (ID) of the broker in this configuration. In the item configuration (see further down in the text) we refer to this ID to establish the communications. Port 1883 is used by default for MQTT traffic.

Openhab uses three configuration files to describe the system:

- *Items* in the directory */etc/openhab2/items;* this contains the definitions of the parameters used, and their connection to the IO.
- *Sitemap* in the directory */etc/openhab2/sitemaps;* this describes the layout of the user interface.
- *Rules* in the directory */etc/openhab2/rules*; this defines which



```
// RFM69 DHT node sketch
//
// This node talks to the MQTT-Gateway and will:
// - send sensor data periodically and on-demand
// - receive commands from the gateway to control actuators
// - receive commands from the gateway to change settings
//
```

operations need to be carried out (scripting).

As en example we'll show the configuration for a DHT end node with nodeID 2, which has a node configuration where the temperature, humidity and output state are periodically transmitted. We create a file called *home.items* in the item directory (*/etc/openhab2/items*), with the contents shown in **Listing 1**. The OpenHAB wiki [12] explains how the configuration of an item should be created. We then create a file called *home.sitemap* in the sitemap directory (*/etc/openhab2/sitemaps*), with the contents:

```
sitemap home label="MyHome"

Frame label="Controls" {
       Switch item=OUT2 label="Myhome lamp"
   icon="light" }

Frame label="Sensoren" {
       Text item=TEMP2
       Text item=HUM2
       Text item=RSSI2   }
```

The first line states the name of this configuration (home), so we can load the correct sitemap when we access OpenHAB with a browser or an App. This is followed with a definition to help display the output of node 2, and then a frame where the temperature, humidity and field strength are shown. And finally, we create a file called *home.rules* in */etc/openhab2/rules* with the following lines:

```
//
//  refresh rules
//

rule "refresh RSSI2"   // periodically refresh
   value for signal strength
    when
       Time cron "0 0/1 * * * ?"
   then
       sendCommand(getRSSI2, "READ")
end
```

The cron command is used to transmit a 'READ' MQTT message every minute to device 2 on node 2, to obtain the field strength.

It's now time to try out a few things. Open a browser and navigate to:

```
http://192.168.xx.xx:8080/basicui/app?sitemap=home
```

You should of course put in the IP address that you're using. The result will be shown in a window like that in **Figure 5**. It's possible that not all of the values are available yet (a '-' will be shown in that case), but after a while the first refresh from the node will occur, when that will be resolved.

Debugging

What to do when things don't appear to work properly.

Is MQTT working?

When the MQTT connection between the gateway and the broker is working, the MQTT led will be lit. This means that messages from the gateway are received by Mosquitto. The next step is to install an MQTT test client (MQTT.fx [13] or MQTTLens [14] from the Chrome web store). MQTT.FX has a *broker status* tab, which can be used to request information about Mosquitto. When you subscribe to the topic *home/rfm_gw/#*, you should see some messages coming through. If all is well, that

Listing 1. OpenHAB configuration for a temperature/humidity sensor end node.

```
Number RSSI2 "RSSI [%d dBm]" {mqtt="<[mosquitto:home/rfm_gw/nb/node02/dev02:state:default]"}
String getRSSI2 "get RSSI" {mqtt=">[mosquitto:home/rfm_gw/sb/node02/dev02:command:*:default]"}
Number TEMP2    "Temperatuur [%.1f °C]" {mqtt="<[mosquitto:home/rfm_gw/nb/node02/dev48:state:default]"}
Number HUM2    "Vochtigheid [%.1f %%]"{mqtt="<[mosquitto:home/rfm_gw/nb/node02/dev49:state:default]"}
Switch OUT2 "zolder node" <PushBtn> {mqtt=">[mosquitto:home/rfm_gw/sb/node02/
    dev16:command:ON:ON],>[mosquitto:home/rfm_gw/sb/node02/dev16:command:OFF:OFF]"}
```

will also include messages from the end node. If that isn't the case, it's time to compile the gateway or the node with the DEBUG option, and to connect a PC to the serial port. Start PUTTY [15], select the required COM port and set its speed to 115200 bps. The output of the node/gateway will indicate if there is a connection and what values are sent. If you use the MQTT client to send commands to the node (for example, send *READ* to */home/rfm_gw/node02/dev03*), you should be able to find out where the data stream fails.

Is OpenHAB working?

If the MQTT connection is working satisfactorily, the next logical step is to look at OpenHAB. A good way of doing this is to inspect the log files. The command

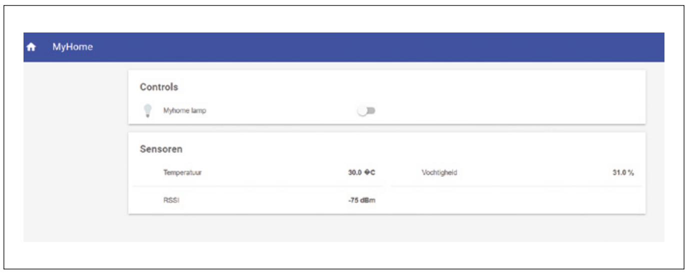
```
> tail -f /var/log/openhab2/events.log
```

will show the last ten lines of the event log. Any updates received from a node or a user interface should be visible here. You should also have a look at */var/log/openhab2/openhab.log*. This is where the OpenHAB system events are logged.

And next...

For the first enhancement you may well be looking at accessing the system via the Internet. This is something you can do via *myopenhab*. Openhab has an app that communicates with your own OpenHAB system via a secure gateway over the Internet. More details can be found at [1].

Openhab also has the facility to show graphs of the variation of measurements over time. This requires that a local database is set up first, along with the selection of measurements that should be stored. This can be done locally, although external databases are also possible. This is described in more detail in [16].



Myhome lamp

Temperatuur 30.0 °C
Vochtigheid 31.0 %
RSSI -75 dBm

Figure 5. Browser window.

The node described here measures the temperature and humidity, and can turn on a digital output. There are many more possibilities when you adapt the electronics in the node, and the software. On Github [3] they show how to connect an LCD and an RFID reader [17]. A node is also described that can control the Dutch range of "klik-aan-klik-uit" switches via OpenHAB. An interesting variation to this project is one based on the ESP8266. This node uses WiFi to create the connection with the MQTT broker and Openhab. The design for this is described at [18].

It's worth having a look on the forum [19] for other applications, where various users have presented designs for the control of PIR detectors, roller shutters, level sensors and more. ◀

(160318)

Web Links

[1] www.myopenhab.org

[2] www.elektormagazine.com/150085

[3] https://github.com/computourist/RFM69-MQTT-client

[4] https://github.com/computourist/RFM69-MQTT-client/blob/master/DHT%20end%20node/RFM_DHT_node_22.ino

[5] https://github.com/LowPowerLab

[6] https://github.com/computourist/RFM69-MQTT-client/blob/master/Gateway_2.4/RFM_MQTT_GW_24.ino

[7] https://github.com/knolleary/pubsubclient

[8] http://harizanov.com/2012/04/rfm12b-and-arduino-ethernet-with-wiznet5100-chip

[9] http://docs.openhab.org/addons/bindings.html

[10] http://docs.openhab.org/installation/openhabian.html

[11] https://sourceforge.net/projects/win32diskimager

[12] https://github.com/openhab/openhab1-addons/wiki/Explanation-of-items

[13] http://mqttfx.org

[14] https://github.com/sandro-k/MQTTLensChromeApp

[15] www.putty.org

[16] http://docs.openhab.org/configuration/persistence.html

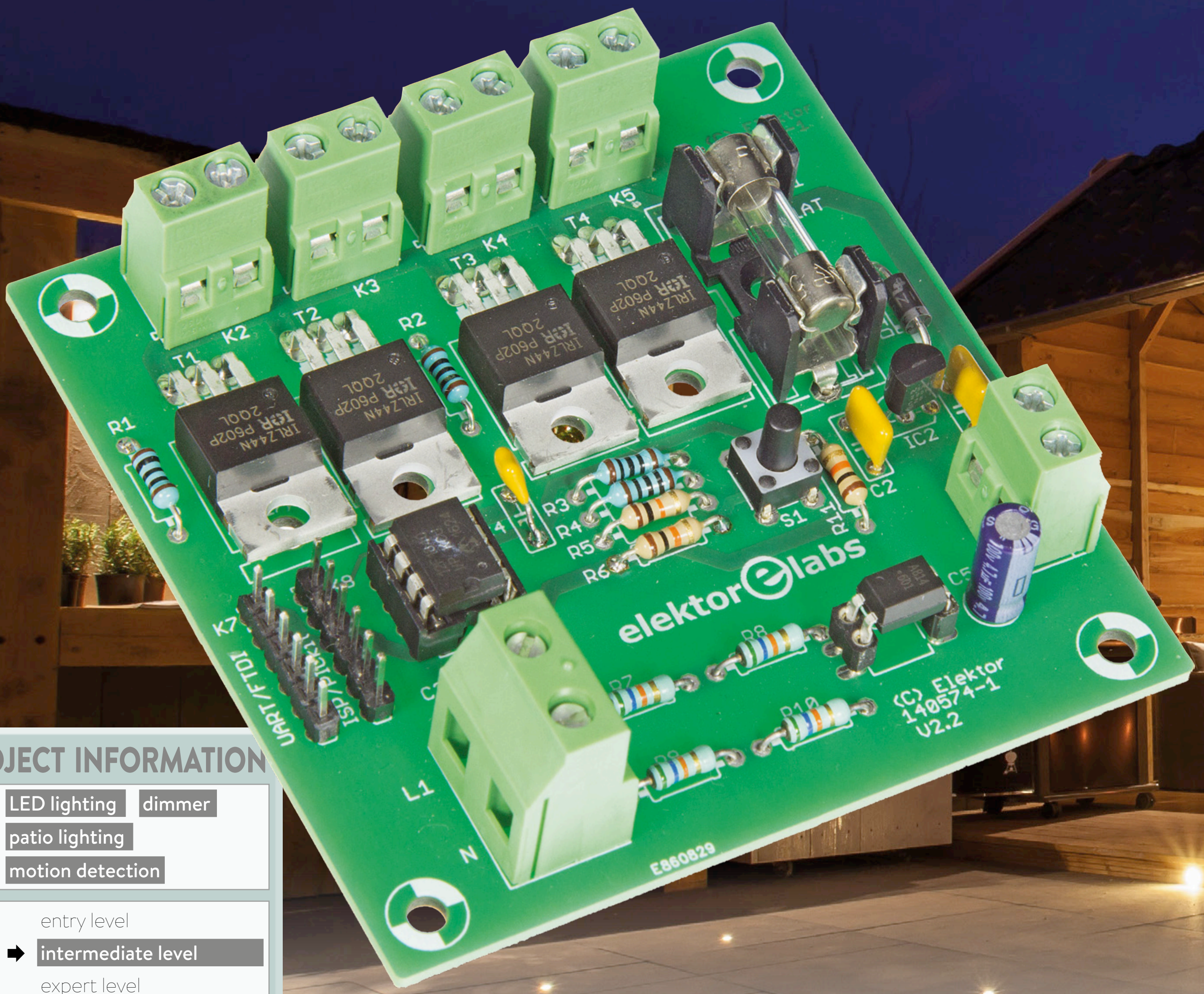
[17] https://github.com/computourist/RFM69-MQTT-client

[18] https://github.com/computourist/ESP8266-MQTT-client

[19] http://homeautomation.proboards.com/board/2/openhab-rfm69-based-arduino

LABS PROJECT

Dimmable Outdoor Lighting



PROJECT INFORMATION

LED lighting | dimmer | patio lighting | motion detection

entry level
➡ intermediate level
expert level

3 hours approx.

FTDI-USB/serial cable
PIC MicroBasic demo version
Visual Basic 2015 (all optional)

€25 / £22 / $27 approx.

By **Andreas Meyer** (Germany)

One bright day last summer our reader Andreas Meyer installed a patio lighting system, using four 3-watt LED spots. The result was far too harsh and bright, however. As nobody expects to wear sunglasses when sitting out on the patio on warm summer evenings, the new lights saw little use. What he needed was a dimming device that would reduce the brightness soothingly and provide greater luxury and convenience...

Customizable brightness settings for 12-V LED lamps

With lighting there's a cast-iron rule that applies to everyone, not just to electronicists: *Any lamp that's far too bright needs a dimmer to make it right!* Unfortunately the four LED spots installed by Andreas had not been intended for dimming. That's not to deny that some examples using 12-V power work fine like this (but hardly ever does this apply for LED lights powered from AC sources). A small test using a PWM signal from a microcontroller with current amplification by power MOSFET (**Figure 1**) indicated that his spots could be dimmed very effectively, even though this was not mentioned in the specification. The reason for the different behavior of 115 V [nominal, North America] or 230 V [nominal, other countries] and 12 V examples is that LED lamps for AC operation generally include a small switchmode power supply, which turns the AC volt-

Characteristics

- Control of four LED lamps
- Two-stage dimming of low-voltage LED lamps
- Smooth transition between brightness levels
- Dimming level controlled by motion detector

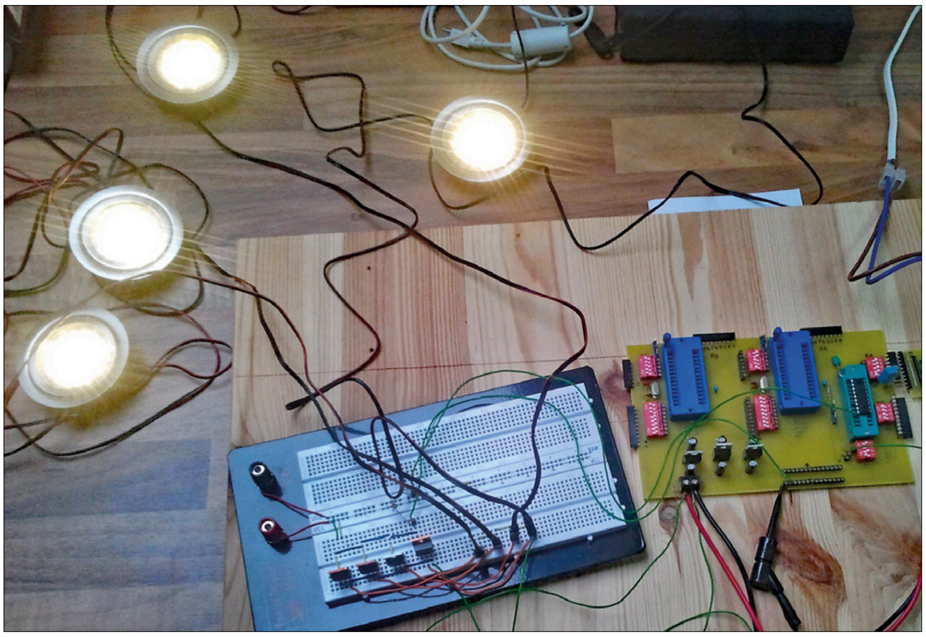


Figure 1. Rapid prototype construction by the author: programmer with microcontroller and four MOSFETs on a breadboard.

Perfect lighting for warm summer evenings!

age into a suitable constant current for the LEDs. This module must of course be designed explicitly for dimming and be able to reduce the output current proportionally to the mean input voltage. Otherwise the AC PSU will attempt to maintain the constant current as long as it can, and when it can no longer do this the lamp will darken, flicker or release an ominous 'electrical' smell. On the other hand 12-V lamps are very often more simple to put together. Frequently you have three white LEDs wired in series, equipped with a suitable dropper resistor, so that a reasonably constant current flows in the 12 V DC voltage. Since there is no electronic control here, dimming achieved by simple variation of the mark/space pulse relationship is not a problem. So much for starters and theory.

Circuit criteria

With decision already taken to base construction around a microcontroller, it was clear that a chip of this kind was guaranteed a place in the finished circuit. But first we had to define our requirements: our experience with the new but barely used patio lighting made it clear that the brightness level should be well below 'full on'. Furthermore, the setup should be capable of activation either by an off-the-shelf twilight switch or a normal power switch, as we didn't want it to switch itself on during the day and, even at night, only when desired. Additionally, it should come on brightly for a short while whenever somebody moved (because we didn't want anybody to

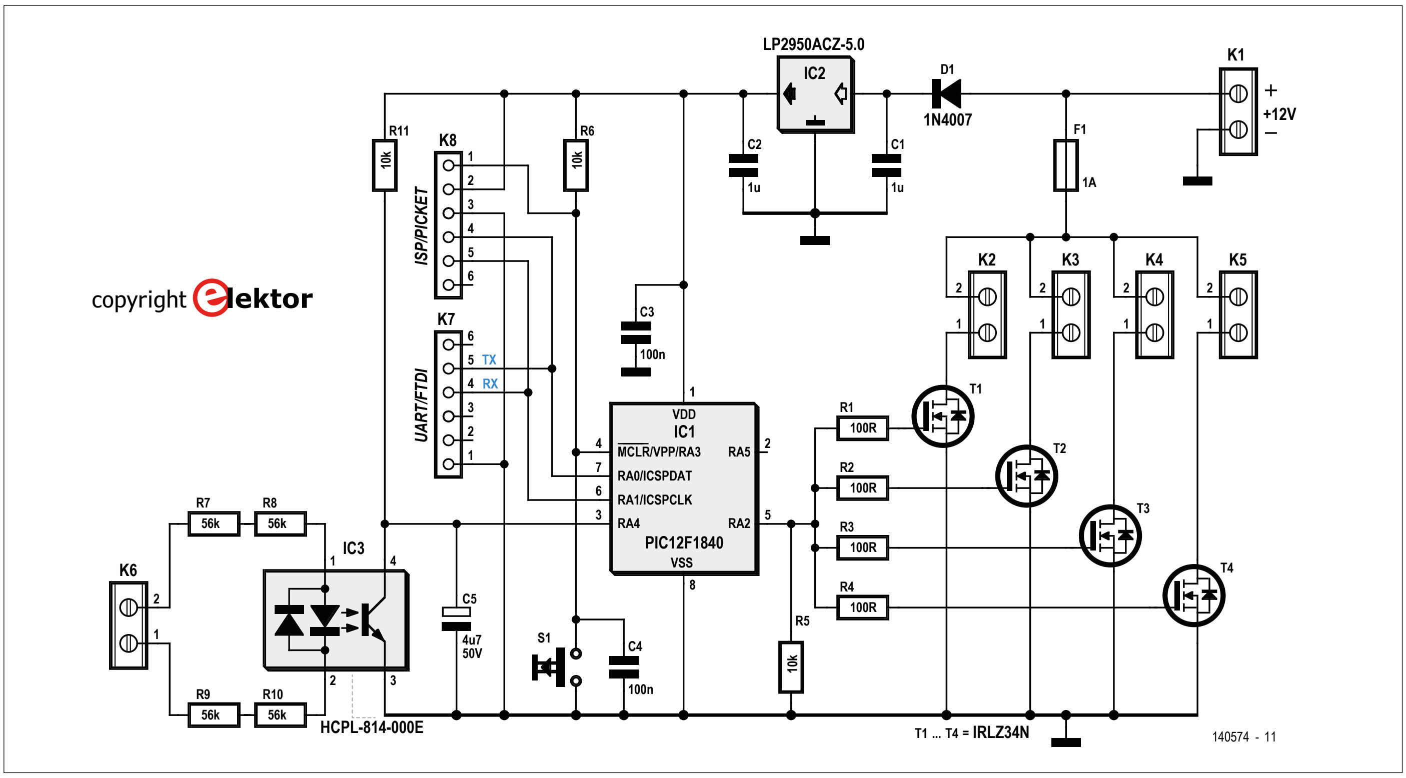


Figure 2. The schematic for dimmable outdoor lighting comprises a microcontroller, four power MOSFETs, a 230 V input for a motion detector and a 5 V voltage regulator.

lose their footing when going to fetch more beer, wine or water). Last but not least, the change in brightness levels should not alter abruptly but gently for convenience.

The dimming function itself using PWM and an MCU was clearly not a problem. You can buy twilight switches as required and simply connect them in series with the light switch. For motion detection you add a suitable movement sensor — you'll find these in every Home Improvement store. The electronics do not require a brightness sensor but a 115/230 V input is needed for the motion detector, so that the lighting can switch between the two desired brightness levels. Speaking of which, the author found the following PWM duty-cycle ratios effective: 10% for normal mood lighting and 50% after movement was detected. The electronics require four PWM outputs plus an AC PSU for powering the 12 V LED lamps and circuitry. If you don't wish to alter the constants in the source code (and recompile/re-flash them into the MCU) every time you adjust the brightness values, it makes a lot of sense to be able to reset the PWM values subsequently with your PC (which will require a serial interface). The main requirements for the circuitry are now fixed.

Actual circuit

Originally Andreas designed the circuit around a PIC16F628 microcontroller that he had handy and used a not particularly sensitive opto-isolator as 230 V control input for the motion detector. Following some feedback from Elektor Labs [1] and subsequent fine-tuning in the labs, the controller in the final version was replaced by the smaller and cheaper PIC12F1840 type that turned out to be fully adequate for the task (see **Figure 2**).

The schematic needs little explanation: at upper right a stabilized 12-V AC PSU capable of supplying at least 1 amp provides the operating voltage. This one amp is adequate for the four 3-W lamps. If using more powerful lights, you'll need a larger AC PSU and fuse F1 to match. The four lamp drivers, in the form of power MOSFETs, will cope with significantly greater current draw even without any heat sinking. IC2, a Micropower voltage regulator, provides 5 V to the microcontroller (a matter of a few mA, no more).

COMPONENT LIST

Resistors
(5%, 0.25W, 250V)
R1–R4 = 100Ω
R5,R6,R7 = 10kΩ
R7 to R10 = 56kΩ *

Capacitors
C1,C2 = 1µF, 50 V, foil, 0.2" pitch
C3,C4 = 100nF, 50V, foil, 0.2" pitch
C5 = 4.7µF, 35V, electrolytic, upright 5x11mm, 2mm pitch

Semiconductors
D1 = 1N4007
T1–T4 = IRLZ34NPBF
IC1 = PIC12F1840-I/P, programmed *
IC2 = LP2950ACZ-5.0
IC3 = HCPL-814-000E, optoisolator

Miscellaneous
5 pcs 2-way screw terminal block, PCB mount, 0.2" pitch
1 pc 2-way screw terminal block, PCB mount, 0.3" pitch
2 pcs 6-pin pinheader, SIL, 0.1" pitch
Fuseholder, PCB mount, e.g. Schurter 0031.8201
Fuse 1A, 20 x 5 mm *
Pushbutton, PCB mount, make contact, 6 x 6 mm
PCB 140574-1 V2.2 *
8-way IC socket, DIP, for IC1

* see text

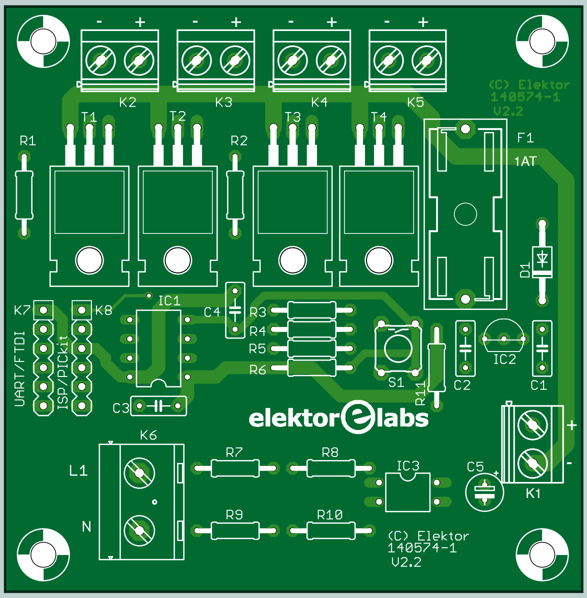


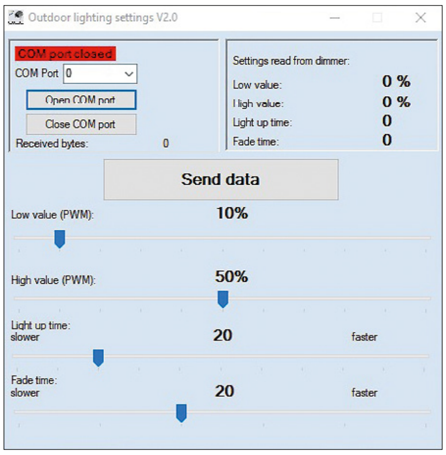
Figure 3. The compact PCB for dimmable external lighting is dimensioned to have a long edge of only approx. 7.7 cms to enable it to fit inside a distribution case of matching size.

When the motion detector is activated, the AC line (mains) voltage connected to K6 is fed via dropper resistors to a sensitive opto-isolator. In this situation we use four 56 kΩ resistors connected in series, not only to share the dissipation loss (around 240 mW in total) across all four but also to enhance survivability in case of component breakdown. C5 on the output of IC3 smoothes the voltage and eliminates any 100 Hz spike pulses in the process. The internal UART of the MCU is connected not only to K7, onto whose header an FTDI-USB/serial cable can be plugged direct, but also to K8, with which you can feed the MCU with software for programming it. Finally S1 provides us with a Reset button.

The PWM frequency is set at 150 Hz by the firmware. For starters, this frequency practically always guarantees flicker-free dimming, and secondly it makes possible a low (internal) clock frequency for the microcontroller of only 200 kHz, with low current draw. Anyone who considers this clock frequency too low can raise it in the firmware, keeping in mind the time constants for the transition between the two brightness levels, which in turn are dependent on the clock frequency of course.

PBC, software and the rest

The PCB designed for this circuit (**Figure 3**) matches the 7.7 cm side length of some cable distribution boxes. As you can see, we avoided the use of SMDs entirely. Thanks to the use of wire-ended components the assembly is straightforward. Besides the PCB, a ready-programmed microcontroller is available direct from the Elektor Store [2]. In its firmware two initial PWM values of 10 % and 50 % are specified, although these can be customized subsequently using your PC and an app of your own. For IC1 it's worth using an IC socket, as given in the component list. As already mentioned, the four MOSFETs do not need cooling. With a typical


Outdoor lighting settings V2.0

COM port closed
COM Port 0
Open COM port
Close COM port
Received bytes: 0

Settings read from dimmer:
Low value: 0 %
High value: 0 %
Light up time: 0
Fade time: 0

Send data

Low value (PWM): 10%
High value (PWM): 50%
Light up time: slower 20 faster
Fade time: slower 20 faster

Figure 4. Screenshot of the window for the PC application used for setting the two dimmer levels and the speed of transition.

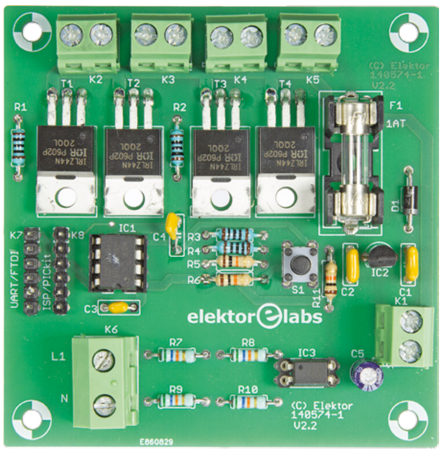


Figure 5. Here's how the prototype built in the Elektor Labs looks.

About the Author

Andreas Meyer is a professional designer in industrial electronics and is passionate about programming microcontrollers and devising circuitry and PCB layouts. Currently he is involved primarily with creating Windows applications that visualize data exchanged with microcontrollers.

'on' channel resistance of 35 mΩ the dissipation loss, even with 1 A current flow, was only 35 mW in each example. And 1 A is adequate for even 4 x 12 W, which generates a huge amount of light when you produce it with LEDs. Incidentally, you should equip the board with only as many MOSFETs as you require output channels. For example, if you are connecting only two lamps, you need two MOSFETs (plus dropper resistors and screw terminals). You can then omit T3 and T4 along with R3 and R4 as well as K4 and K5. It goes without saying that if you plan any total output higher than 12 W you'll need to alter the ratings of F1 and the AC PSU.

The firmware of the PIC12F1840 was developed using the mikroBasic IDE from MikroElektronika. Since the code turns out so compact thanks to the modest number of functions used, it can even be adapted and altered using the free demo version of this IDE. What's more, Andreas Meyer has also developed a small PC app for Windows with the aid of Visual Studio (see **Figure 4**), with which the two PWM values for the two brightness values can be transferred to the microcontroller using an FTDI-USB serial converter cable. Another tip for customizations of your own: changed values are implemented by the MCU only after a Reset! The entire source code, the ready-to-use PC program and the Hex file for the firmware are of course available for free download at the Elektor web page for this project [2]. **Figure 5** shows the prototype constructed in the Elektor Labs.

Wait, there's more...

Do not even think of procuring one of those low-cost 'electronic 12 V transformers' in rectangular white plastic boxes, as sold for use with halogen lamps for instance. This is categorically not the way to go. Those devices generally contain a crude switchmode AC PSU, whose output is indeed 12 V but supplied as an AC voltage alternating somewhere in the tens of kHz. A power supply like this is fine for incandescent bulbs but would do LED lamps no good at all (use the circuitry described in this article instead!). The MOSFETs all contain integral parasitic diodes connected in antiparallel, as made clear by the circuit symbol. In this way the LED lamps you connect always receive at least a half-wave of the AC voltage, and if the dimming range is reduced to between 50% and 100% and AC actually reaches the LEDs, this does not increase their life unconditionally, since LEDs are only designed for low reverse voltages.

The second point to note is the installation itself: when the lamps are connected there is a squarewave voltage at the four screw terminals K2 to K5 whose frequency corresponds with the PWM. Although 150 Hz may not sound much, the squarewave voltage has fairly fast rise and fall times and this leads to rich harmonics that extend well into the radio frequency ranges. In other words, unless the wiring to the LED lamps is kept very short, you'll be transmitting radio interference into the ether and across your neighborhood. Standard domestic wiring using normal cables or single conductors made of rigid wire are not designed with RF in mind, so they are not 'RF immunized'. In such situations, twisted bell wire is a bit better, and the use of shielded cable is even better for wiring extending more than a few meters. Both bell wire and normal RG-58/U coaxial cable can tolerate 2 A easily; they may not be intended for our purpose but will minimize unwanted radiation from low-voltage cabling of our kind.

If, like Andreas Meyer, you enhance your dimmable outdoor lighting with motion detection (that is with two levels of dimming), you may well need to adjust the two brightness settings several times after installing the lamps. This makes it advisable to finalize constructing the circuit only after you have found the optimal compromise for eventide lighting — or at least provide easy access to K7 for connecting a PC or Laptop (or to K8, in case you plan to upgrade the firmware with new values). This will simplify the optimization task substantially. And then you're set to enjoy those balmy summer evenings... ⏮

(140574)

Web Links

[1] http://goo.gl/izezWr

[2] www.elektormagazine.com/140574

FROM THE STORE

→140574-1
PCB

→140574-41
Programmed microcontroller

Presence Simulator

Discourage the burglars

By **Theo de Wijs** (Netherlands)

Members of the burglars' guild (part of the *genus* 'sticky-fingered') have a particular interpretation of 'mine' and 'yours', which doesn't have much public support. They always arrive unannounced and when it comes down to it, the thing they're most discouraged by is the presence of the rightful owners of the home and all stuff inside.

The best way to prevent a burglary would be to stay at home permanently. This is of course not practical for obvious reasons, and longer absences (during the holidays, for example) provide more opportunities for burglars to pay you an unwelcome visit. We need to do something about this, and it can be done in two ways. The first option is to have an alarm system installed, but this can easily get very expensive. Discounters sometimes offer cheaper installations, but these systems often have some essential functions missing when you compare them with other systems, so they're not really an option. Furthermore, you could argue that it's too late when the alarm goes off, since the damage will already have been done...

The other option is to simulate a continuous occupation of the home, which is what this project is about. We obviously won't use a simple timer circuit that switches a light on and off at the same times every day, since any self-respecting criminal will spot this straight away. We therefore have to put a bit more effort into the circuit. The circuit presented here reacts to the diminishing daylight in the evening to close a roller shutter. A lamp or a TV simulator is turned on soon afterwards, and turned off again later (at 'bedtime'). In the morning the roller shutter is raised again.

Circuit diagram

When we look at the circuit diagram in **Figure 1** from left to right, we come across its various sections. At the far-left we have the power supply. The supply voltage is derived from the AC line via R1 and C1, then rectified by D1-D4 and stabilized at 12 V by D5/C2.

Be aware that the complete circuit is connected to the AC line voltage!

This not only means that the circuit must be constructed soundly and be fully isolated, but also that careless experiments and measurements could be **deadly**.

Directly to the right of the supply is a

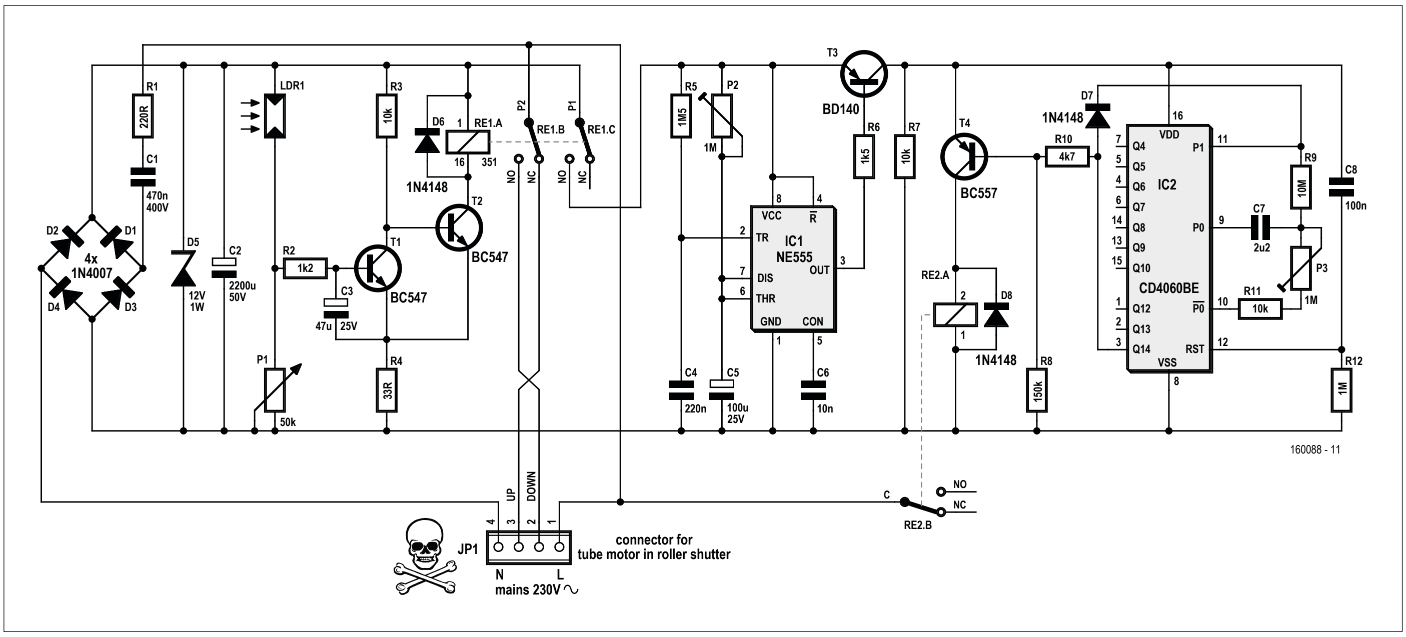


Figure 1. The circuit consists of three sections: power supply, sensor and driver.

light dependent potential divider, using LDR1 and P1. As long as (sufficient) light falls on the LDR during the day, its resistance is low and the voltage at the junction with P1 is high. This means that transistor T1 conducts, resulting in a low voltage at its collector. Transistor T2 will therefore be off and no current flows through the coil of RE.1A, so the circuit will be at rest. The AC line voltage is connected to the UP pin of JP1 via the NC contact (*Normally Closed*) of the double pole relay (RE.1). The roller-up connection of the roller shutter tubular motor is connected to this, causing the roller shutter to open and stay that way. In this state the supply voltage is *not* connected (via RE1.C) to the rest of the circuit around IC1 and IC2.

When the light level drops during the evening, the resistance of the light dependent resistor will increase. This causes the voltage at the junction of LDR1 and P1 to drop and at a certain point this voltage will no longer be high enough to keep T1 in conduction (the exact moment when this happens can be adjusted using P1). Once T1 is blocked, its collector will be high and T2 conducts. This causes a current to flow through the coil of RE1.A, the relay engages and the circuit is active. The AC line voltage is now routed to the DOWN pin of JP1 via RE1.B, the similarly named connection of the roller shutter motor now gets power, and the shutter closes.

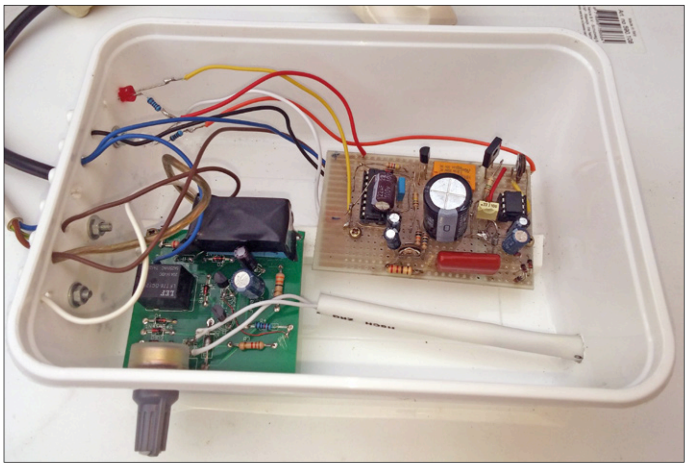


Figure 2. The prototype of the author has an enclosure which is 'open to improvement'.

The Author

Theo de Wijs (73 years old) is a retired executive of a support organization. He took an electronics course during the time of vacuum tubes and the first transistors. He was employed in the electronics sector up to 1976 (with Philips, amongst others); he then trained to work in support organizations. Electronics has always remained his hobby.

Timers

The second part of the circuit is also powered now via the NO contact (*Normally Open*) of RE1.C: timer IC1 (the good old NE555). This timer has a very important function. It would be noticeable if a lamp (or anything else) were turned on at the exact moment when the roller shutter started closing. Criminals would likely come to the conclusion that both events happened suspiciously close together. This would therefore negate most of the usefulness of the simulator.

When timer IC1 is first supplied with power, its output (pin 3) will be high. PNP transistor T3 will then be in a blocked state, meaning that the second timer (IC2) won't get a supply voltage. After the monostable period (adjustable via preset P2) of IC1, which should be between about 30 to 50 seconds, the output on pin 3 goes low and T3 provides power to the second timer (IC2).

Strictly speaking, IC2 (an 4060) isn't a 'real' timer, but a digital counter with an integrated oscillator circuit and a large number of counter outputs. When the supply volt-

TV simulator

A TV simulator has been mentioned several times in this article. This is a simple circuit with a number of differently colored LEDs and a (pseudo) random number generator that simulates the flickering light of a color TV. These devices can be bought cheaply.

age is turned on by T3, the reset input of the IC (pin 12) gets a pulse via C8/R12, which turns all Q outputs of the IC to their low state, including output Q14 (pin 3). PNP transistor T4 is then turned on and supplies a current to relay RE2.A. This engages so that the AC line voltage becomes available at the NO contact to power a lamp or a TV simulator (or anything else).

After a certain period, which can be set within a wide range (up to a maximum of nearly 24 hours) using preset P3, output Q14 goes high. This causes T4 to turn off, and hence turns off relay RE2.A as well. This is the simulated bedtime: the TV simulator and/or lamp are turned off. Diode D7 plays an important role in this timer: it stops the oscillator of the IC as soon as Q14 goes high, and prevents the counter from continuing.

The timer circuit is now in a stable state: counter IC2 has stopped, relay RE2.A has turned off, and that remains the same until the next day when the sun rises again (at least, we hope that's the case). More light will then fall on the LDR, the voltage at the junction of the LDR1 and P1 rises, and transistor T1 conducts. T2 then blocks, relay RE1.A turns off and the circuit is at rest. IC1 and IC2 no longer get a supply voltage, which resets the timer circuit so it is ready for when the next cycle begins.

A few more details

This just about completes the description of the circuit. The circuit has been designed so that both relays are powered up while it is dark. This seemed the best way to do it, since we usually go on holiday during the summer, when the nights are shorter than the days. If you want to use this circuit during the darker season (Winter sport fanatics take note), you could use a PNP device for T2 and swap the connections to the relay. The relay will then be turned off when it is dark, reducing the power consumption of the circuit somewhat.

It is important that the tubular motor used for raising and lowering the roller shutter is provided with end-stops. In other words: when the shutter becomes either fully raised or fully lowered, the power to the motor should be disconnected by the end-stops (since the AC line voltage supplied by relay RE1.A is continuous). Luckily, this happens to be the case with most roller shutters. The relay contacts must obviously be able to deal with the currents they have to switch. For RE2 this won't be a problem; when selecting RE1 we would recommend that you first read the documentation for the roller shutter. However, a 10A device should be sufficient in most cases.

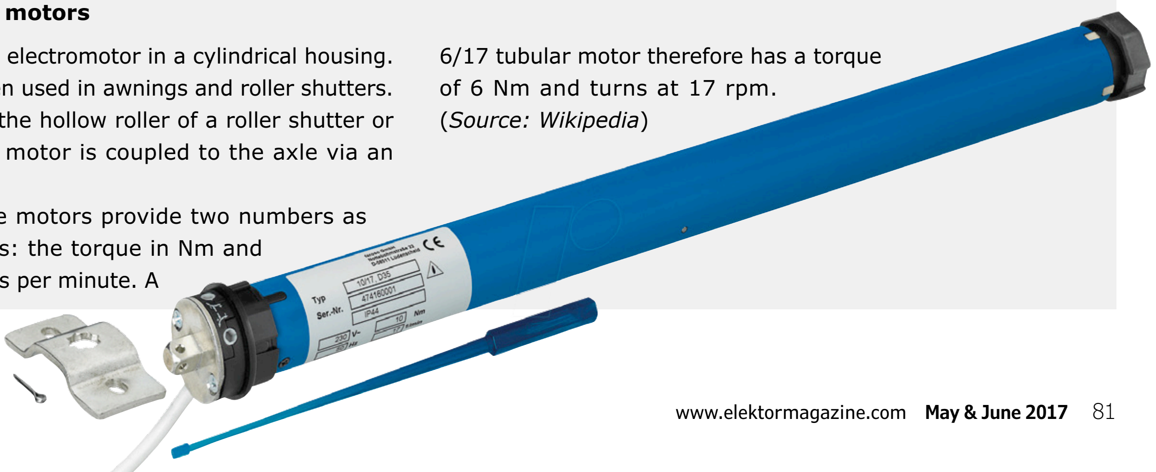
At the start of this article we mentioned that the circuit is connected directly to the AC line, without using a transformer for the power supply. This shouldn't be a problem, although you must pay more attention to the construction than usual. The circuit must be housed in a strong, plastic enclosure so that none of the internal parts can be touched; this applies particularly to potentiometer P1 and the LDR: you must use one that has a plastic spindle!

Figure 2 shows the prototype of the author. Part of the circuit was built on experimenter's board. Although the whole circuit was originally put together quickly as a proof-of-concept, it has been in use like this for quite some time. Nevertheless, we would recommend that the circuit is constructed in safer way and in a better enclosure if you intend to leave it running unattended for several weeks at a time. ◀

(160088)

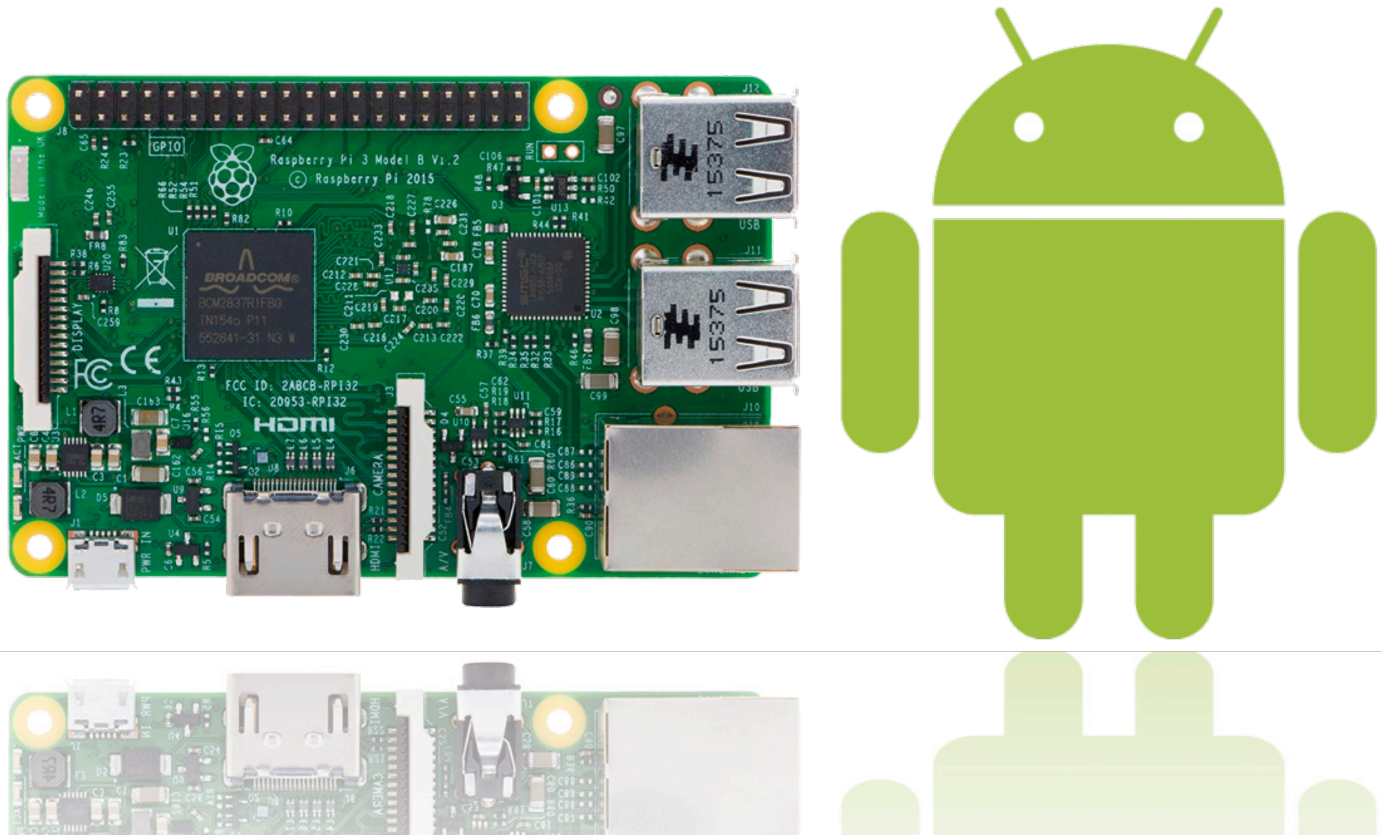
Talking about tubular motors

A tubular motor is a long electromotor in a cylindrical housing. This type of motor is often used in awnings and roller shutters. They're mounted inside the hollow roller of a roller shutter or an awning. The tubular motor is coupled to the axle via an adapter and a catch.
All manufacturers of the motors provide two numbers as part of the specifications: the torque in Nm and the number of revolutions per minute. A 6/17 tubular motor therefore has a torque of 6 Nm and turns at 17 rpm.
(*Source: Wikipedia*)



Android on your RPi (2)

Controlling a display using SPI



In our previous edition we got our first Android program running. In it we switched an output between High and Low in an endless loop in order to investigate its behavior in real time [1]. If we now wish to control peripherals, operating Pins 'manually' does not make sense — instead we use the Hardware Bus Interface of the Controller. In this article we show you how to control a small OLED display using SPI.

By **Tam Hanna** (Slovakia)

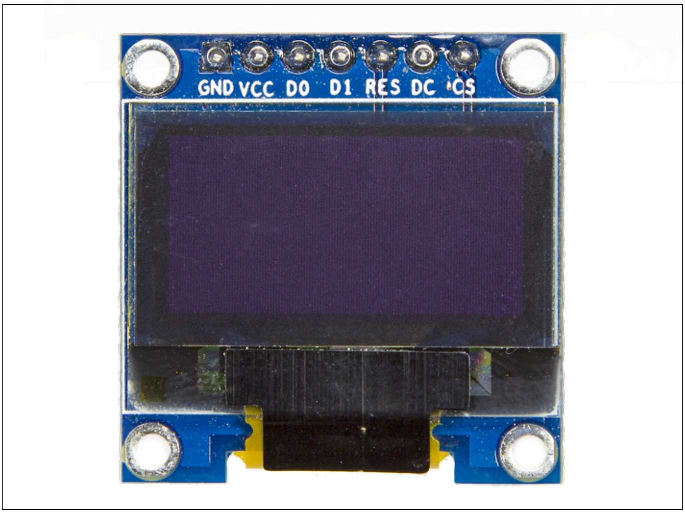


Figure 1. in terms of both current consumption and size this display is significantly smaller than an HD monitor.

If you plan to control peripherals with Android Things you are currently spoilt for choice between SPI and I²C. In the steps that follow we will use one of the widely available SPI OLED displays such as the part shown in **Figure 1**, which at time of writing is available from aliexpress.com for around $2.50 with free shipping (and a long wait) from China.

Circuitry

From a hardware perspective a Raspberry Pi with Android behaves just like a regular RPi. The circuit shown in **Figure 2** is very similar to the one used in the series *Windows on the Raspberry Pi* [2].

Android Things identifies SPI buses (just as it does GPIO Pins) by using Strings. Our first exercise consists of identifying those available on the Raspberry Pi:

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    . . .

    PeripheralManagerService manager = new
         PeripheralManagerService();
    List<String> deviceList = manager.getSpiBusList();

    if (deviceList.isEmpty()) { } else {
        Log.i(TAG, "List of available devices: " +
              deviceList);
    }
}
```

The program generates a list of all buses available in the Debugger Console. At the time of writing the author's Raspberry Pi 3 indicated two endpoints:

```
I/MainActivity: List of available devices: [SPI0.0,
      SPI0.1]
```

At this stage Gradle and Android Studio produce a minor trap for the unwary: if you plan to perform data extraction using a Breakpoint, you must be sure to start the program by clicking on the Debug Symbol. If you click on the normal Run arrow, the Debugger will not be embedded with the program.

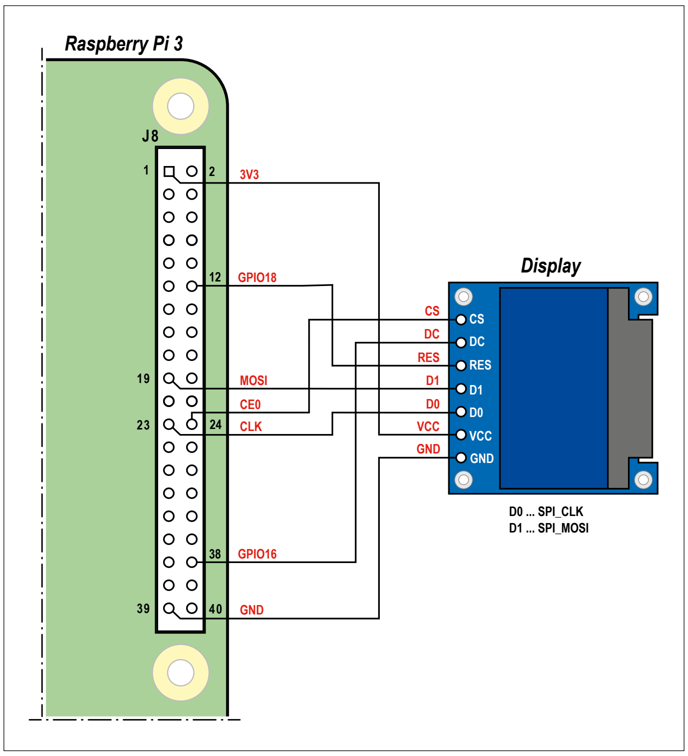


Figure 2. A handful of wires link the Raspberry Pi and the display module.

When creating a fresh demo project our first task is to initialize the SPI Bus (it goes without saying that you can still download the code again from the Elektor website [3]). For this we use a new Method named `configureAndFindDevice`. First up we expand the Main Activity with a global GPIO Object, which controls the Reset connection on the display. We then apply an SPI device that encapsulates the actual communication between the display and our Pi.

```
public class MainActivity extends Activity {

    SpiDevice myDevice;

    Gpio myPinReset;
```

The following step covers the — more or less classic — means of initializing the SPI Bus within `configureAndFindDevice` (**Listing 1**). Invoking `setFrequency` specifies the communication frequency — in the author's tests both 4 MHz and 8 MHz worked fine without any problems.

Attentive readers who bothered to look at the listing may possibly wonder what `Thread.Sleep` in the GUI Thread is searching for. The answer is that a delay of 10 ms is of no consequence and accordingly does not justify the effort of generating a separate Thread.
To invoke the Method we use `onCreate`, so that it takes place at the start of the Program (**Listing 2**).

Next we initialize an additional GPIO Pin to facilitate control of the Data Control line. We also invoke the `initDisplay` Function, which is responsible for defining the various parameters of the displays (the content of this Method follows in the next section). For the sake of total completeness we should note that the SPI device needs to be released when the Activity is ended — this is handled using the code in `onDestroy`, see **Listing 3**.
If these *close* commands are missing, during Debugging

Listing 1. Initializing the SPI interface.

```
void configureAndFindDevice(){

    PeripheralManagerService manager = new
                PeripheralManagerService();

    try {

        myPinReset = manager.openGpio("BCM18");
        myPinReset.setDirection(Gpio.
            DIRECTION_OUT_INITIALLY_LOW);
        myPinReset.setValue(false);

        myDevice = manager.openSpiDevice("SPI0.0");
        myDevice.setMode(SpiDevice.MODE0);
        myDevice.setFrequency(4000000);
        myDevice.setBitsPerWord(8);
        myDevice.setBitJustification(false);
        Thread.sleep(10);
        myPinReset.setValue(true);
    }

   catch (Exception e) {. . .}

}
```

Listing 2. Initializing the pins uses the method *onCreate*, invoked when the program starts.

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    . . .

    configureAndFindDevice();

    try {

        PeripheralManagerService manager = new
                PeripheralManagerService();
        myPinDataCommand = manager.
                openGpio("BCM16");
        myPinDataCommand.setDirection(Gpio.
                DIRECTION_OUT_INITIALLY_LOW);

    }catch(Exception x){}

    initDisplay();

}
```

you may occasionally meet Exceptions of the type `android.os.ServiceSpecificException: BCM18 is already in use`. Note too that it is also permissible to remove Activity Classes of the Android OS without interrupting an ongoing operation — whether you do this is up to your own personal fancy, which frequently causes strife in the world of Android.

Let initialization begin...

Having successfully configured the SPI Framework we can move forward to data transfer, in order to deploy the display controller into the correct mode and bring data onto the monitor screen. The data sheet for the SSD1306 is available widely but when using an IC for the first time it is also a good idea to check out the ready-to-use Libraries at Adafruit and in the documentation of various other manufacturers. These will show you how the chip is used in practice.

You can see the code for `initDisplay` in **Listing 4**.

Next we invoke `Thread.sleep` once more. This ensures that the Controller has sufficient time to start up properly following arousal from Reset status. After this we send three Commands; these look after activating the charge pump and switching on the display. Before invoking the command `sendData` (responsible for transmitting data for the display), we also need to make sure that the display Buffer contains some more or less random information.

The actual declaration of the Command Constants follows a somewhat strange procedure. It is interesting to note that before each individual Constant, a 'Cast' Byte is required. If this is missing, the Java compiler would otherwise set up a Tree:

```
private byte[] CMD_DISPLAY_ON = { (byte)0xAF };
```

Next, we can turn to the two methods that send information to the display controller. First a brief explanatory note: the SD1306 works as an ordinary SPI device, both when accepting data and when collecting commands. The level at the digital DC input determines the mode. If the DC signal is high during the transmission of data, the data is moved into the display

Listing 3. Tidying up after the program ends.

```
@Override

protected void onDestroy() {

    super.onDestroy();

    Log.d(TAG, "onDestroy");

    try {
        if(myDevice!=null) myDevice.close();
        if(myPinDataCommand!=null) myPinDataCommand.
              close();
        if(myPinReset!=null) myPinReset.close();
        myDevice = null;

    } catch (IOException e) {

        Log.w(TAG, "Unable to close SPI device", e);

    }

}
```

Listing 4. Displaying lines on the screen.

```
void initDisplay()

{

    try {

        Thread.sleep(100);
        SendCommand(CMD_CHARGEPUMP_ON);
        SendCommand(CMD_MEMADDRMODE);
        SendCommand(CMD_DISPLAY_ON);

        for(int i=0;i<myDisplayBuffer.length;i++){
            myDisplayBuffer[i]=22;
        }

        sendData();
    }

        catch(Exception x){Log.e("Elektor",x.
getMessage());}

}
```

Buffer — if it is low, the information is instead written into the Command memory. This leads to the following code:

```
    void SendCommand(byte[] _what){

        try {
            myPinDataCommand.setValue(false);
            myDevice.write(_what, _what.length);
        }

        catch (Exception x){
            Log.e("Elektor",x.getMessage());
        }
    }
```

The version for transferring display data differs only minimally from `SendCommand`. Instead of accepting data from the Function Parameter, the Function serves as a Member of an Array (`myDisplayBuffer`).

```
    void sendData(){

        try {
            myPinDataCommand.setValue(true);
            myDevice.write(myDisplayBuffer,
                 myDisplayBuffer.length);
        }

        catch (Exception x){
            Log.e("Elektor",x.getMessage());
        }
    }
}
```

And with this, the first version of our program is ready to put into use. Try it out now — if you have implemented it correctly, a series of horizontal lines will appear on the screen (**Figure 3**).

Now we'll communicate for real

We don't need a dedicated Real Time operating system to display line patterns; even on an AVR a display memory of just 1 KB can be used successfully for this task. But you can see how the Raspberry Pi shows its superior capabilities when the operating system is displaying complex information.

A genuinely interesting exercise would be to show GUI control elements (for instance a diagram) on the screen as a Bitmap. To do this let's embed *MPAndroidChart* in our program — *MPAndroidChart* is a high performance diagrams Library that also plays a role in the development of classic Android applications. The first step is to open the file *build.gradle* covering the entire project and enhance it using the following passage:

```
allprojects {
    repositories {
        jcenter()
        maven { url "https://jitpack.io" }
    }
}
```

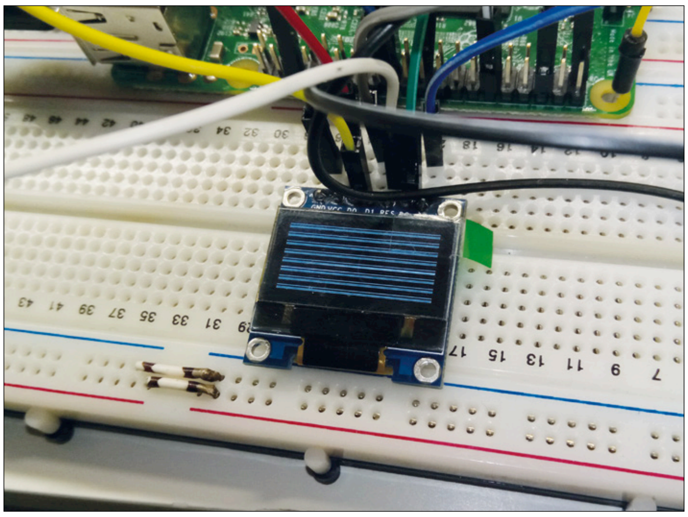


Figure 3. The test pattern confirms that everything is functioning perfectly. On the other hand a completely black OLED indicates it's 'dead'.

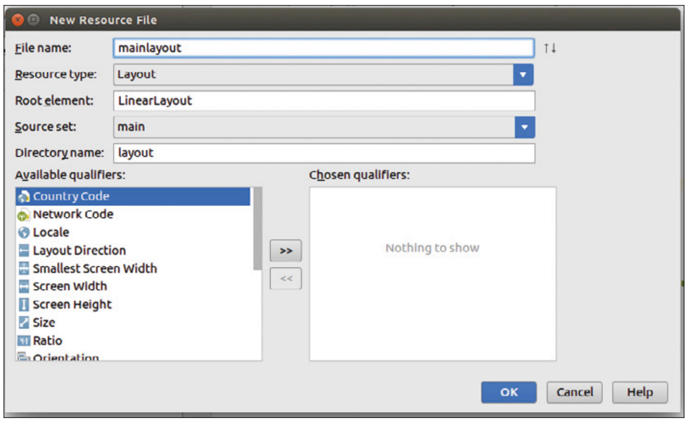
Next we expand the file *build.gradle (Module:app)* relating to the Application as follows:

```
dependencies {
    provided 'com.google.android.
          things:androidthings:0.1-devpreview'
    compile 'com.github.PhilJay:MPAndroidChart:v3.0.1'
}
```

After saving both files Android Studio superimposes a yellow banner, on which you now click on *Sync Now*. The IDE now actualizes the project in its entirety — one of the key strengths of Gradle is its ability to download Libraries and other necessary support materials from the Internet actually *during the compilation process*.

In the next step right-click on the folder for this project and select the option *New ➞ File ➞ Android Resource File*. In the dialog that now appears select the settings indicated in **Figure 4** so as to create a Layout File.

At this stage the Layout Subsystem implemented in Android would normally cause us grief, because it would dynamically



New Resource File

File name: mainlayout
Resource type: Layout
Root element: LinearLayout
Source set: main
Directory name: layout
Available qualifiers:
Chosen qualifiers:

Figure 4. These Parameters produce a new Layout file.

adapt the size of the control elements shown. However, here we need to have a Widget having the size 128 × 64 pixels, which the following adjustments to the code will define:

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android=http://schemas.android.com/
            apk/res/android
    android:orientation="vertical"
            android:layout_width="match_parent"
    android:layout_height="match_parent">

    <com.github.mikephil.charting.charts.LineChart
        android:id="@+id/chart1"
        android:layout_width="128px"
        android:layout_height="64px"/>

</LinearLayout>
```

The circuitous detour via the main screen of the Raspberry Pi employed here is not absolutely necessary but it simplifies our work, as we can then view the results of this exercise on a studio monitor that we might also wish to connect.

In the next step we turn to the Chart Library, which of course requires a bit of help before displaying diagrams. This now looks as follows:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.mainlayout);
```

The first difference from a typical Android Things application is the invocation of `setContentView` — this takes care of loading the information found in the *XML Layout*.
Next we create an Array that delivers more or less random display data:

```
    ArrayList<Entry> values = new ArrayList<Entry>();

    for (int i = 0; i < 20; i++) {
        float val = (float) (Math.random() * 25) + 3;
        values.add(new Entry(i, val));
    }
```

Following this we define the parameter settings: *MPAndroid-Chart* is provided for handling large amounts of data, which is why the Initialization turns out to be relatively complex (and why we show only a few extracts):

```
    LineChart mChart = (LineChart) findViewById(R.
         id.chart1);
    mChart.getDescription().setEnabled(false);
    LineDataSet s1=new LineDataSet(values,"");

    . . .

    ArrayList<ILineDataSet> dataSets = new
          ArrayList<ILineDataSet>();
    dataSets.add(s1);
    LineData data = new LineData(dataSets);
    mChart.setData(data);
```

The primary difficulty, which leads to the enormous length of the code, is the elimination of unnecessary drawing commands. The display resolution of 128 × 64 Pixels is so small that the developers of *MPAndroidChart* probably never even tested this. Our next exercise consists merely of converting the result into a Bitmap that can be sent to the display. Here it is crucially important to discard the color information, because as you know, our screen is only monochrome (black and white).

On account of the comparatively bizarre drawing technique of *MPAndroidChart*, we need to slow down the conversion process. This can be accomplished using a Handler. The bare bones of the code used in the Handler are as follows:

```
final Handler handler = new Handler();
handler.postDelayed(new Runnable() {

    @Override
    public void run() {

        //Code ist hier

    }
}, 700);
```

`postDelayed` accepts an Object that needs to prepare a `run()` Method, in which we have the code whose execution needs to be delayed. This is very extensive, so we'll begin with the conversion of the Widget into an Instance of the Bitmap Class:

```
    public void run() {
        LineChart mChart = (LineChart) findViewById(R.
                id.chart1);

        Bitmap returnedBitmap = Bitmap.createBitmap(128,
                 64,Bitmap.Config.ARGB_8888);
        Canvas canvas = new Canvas(returnedBitmap);

        Drawable bgDrawable =mChart.getBackground();
        if (bgDrawable!=null)
            bgDrawable.draw(canvas);
        else
            canvas.drawColor(Color.WHITE);

        mChart.draw(canvas)
```

At this stage `returnedBitmap` contains a full-color version of the diagram rendered by *MPAndroidChart*. Our next task consists of converting this Bitmap into a Frame Buffer. The first challenge is to construct a bank of monitor screen Memory that is split into a group of 'pages'.
The simplest way of solving problems of this kind is to pick clean the Memory cells, step by step, using *for* Loops. In our case three Loops are necessary to do this:

```
        byte counter=0;
        byte store=0;
        int counter2=0;
```

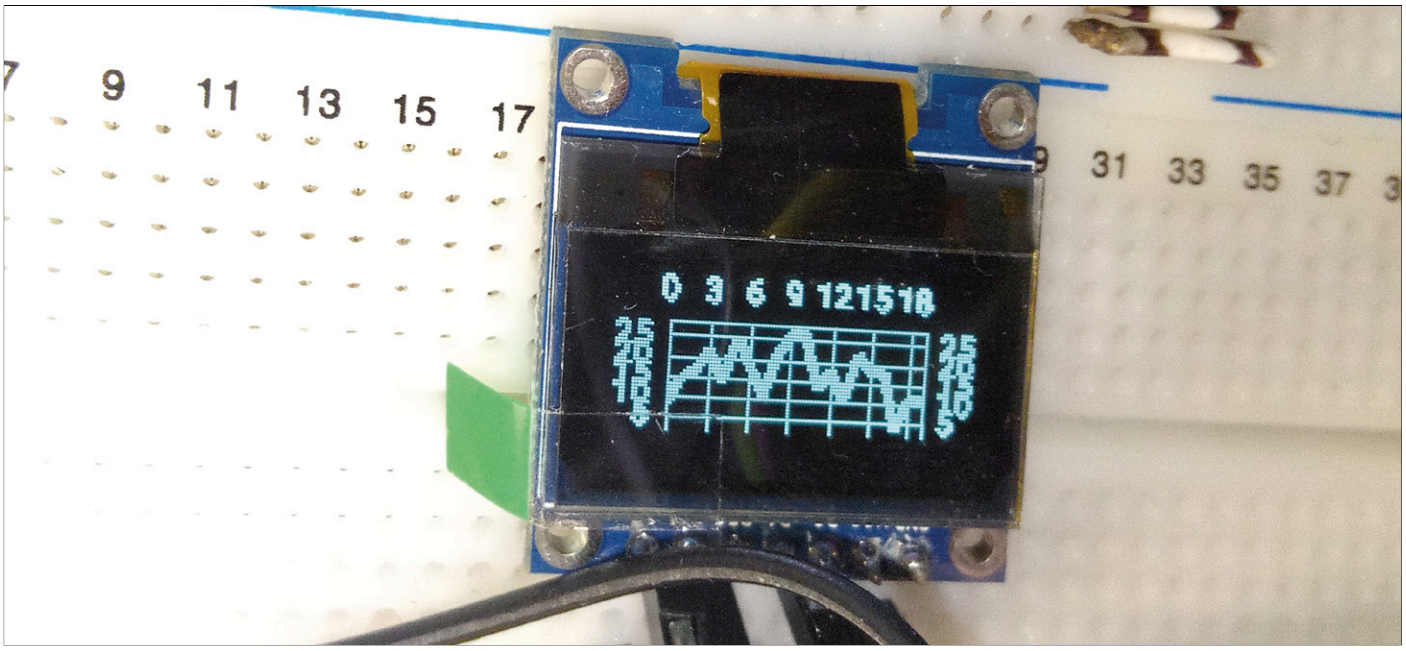


Figure 5. Thanks to Android we can achieve an attractive line graph with modest effort.

```
    for(int ctr=0;ctr<8;ctr++)
        for(int x=0;x<returnedBitmap.getWidth();x++)
    {
            for (int y = ctr*8; y < (ctr+1)*8; y++)
    {
```

Next comes the thankless task of grouping every eight Pixels into a Byte. We start off by procuring the color information of the respective Pixels, so that we can then convert the color components, using simple comparison techniques, into 'black' or 'white':

```
  int myVal=returnedBitmap.getPixel(x,y);
  if(Color.red(myVal)!=255 || Color.
     green(myVal)!=255 || Color.blue(myVal)!=255)
           { //Set
         store=(byte)(store | ((byte)(1 <<
              (counter))));
  }
```

The actual outputting takes place in increments of eight. Using Bit operators, `store` is filled progressively with data — and as soon as an eight-Bit Word is 'complete', it is shifted into `myDisplayBuffer`.

```
                counter++;
                if(counter==8) {
                    myDisplayBuffer[counter2]=store;
                    store=0;
                    counter=0;
                    //Byte raus
                    counter2++;
                }
            }
        }
```

Last but not least, we direct a Command to the Display Controllers to establish a defined condition. `sendData` then passes the data via the SPI Bus to the display:

```
        SendCommand(CMD_RESETCOLADDR);
        SendCommand(CMD_RESETPAGEADDR);

        sendData();

    }
```

The reward for your efforts is the diagram shown in **Figure 5**.

And finally

We've reached the end of our experiments with Android Things. It's a platform that certainly won't appeal to all users, but anyone prepared to get to grips with it will be enriched with an immense range of new capabilities. With classic eight-bit machines many tasks are impossible — or virtually impossible — to bring to reality. Whereas putting this kind of high-performance 32-bit process computer on the job really saves you time and effort. And with this in mind we wish you plenty of success! ⏮

(160369)

Web Links

[1] www.elektormagazine.com/160361

[2] www.elektormagazine.com/150520

[3] www.elektormagazine.com/160369

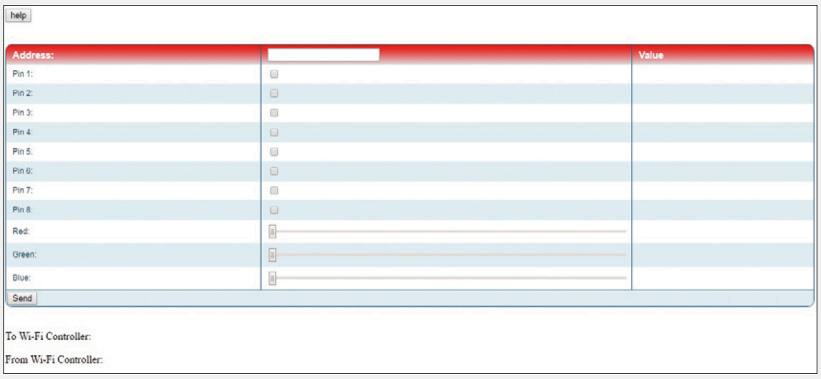
Err-lectronics

Corrections, Updates and Feedback to published articles

Return of the WLAN Controller Board

Elektor 6/2016, p. 94 (150402)

UPDATE. Control from a PC (via the website 150402.html) is currently not possible using the Internet Explorer or Edge browsers. These browsers do not allow local data storage (in this case the IP address entered for the WLAN controller board) when websites are stored on a local hard disk drive. To avoid this problem, Clemens Valens of Elektor Labs has produced a new version of the website. It now queries the browser type and prevents local data storage if that is not possible. However, the data from the form is still sent, so the controller functions properly. The new version of the website is located on the /Labs page at www.elektormagazine.com/labs/remake-wifi-controller-board-150402 and on the project page at www.elektormagazine.com/150402.



D-Watt

Elektor 1/2017, p. 74 (150115)

CORRECTION. In the article it says near the end of page 76: “It is designed with a gain of 1, but if you wish you can increase the gain by fitting a resistor in the R3 position (A = R4/R3 + 1)”. What this means is that with R3 not fitted the gain is unity, but you can increase the gain by fitting a resistor for R3 according to the formula A = R4/R3+1.

D-Watt

Elektor 1/2017, p. 74 (150115)

CORRECTION. The article does not cover the adjustment of trimpot P1. Use the following procedure: After everything is connected, switch on the amplifier and check the PWM frequency of each channel (it should be 400 kHz). Measure the frequency at the junction of L1 and T1 with no input signal, and adjust P1 as necessary to obtain the right frequency. If you do not have a suitable instrument for measuring the frequency, set P1 to the midrange position in both channels. You can read more about this at Elektor/Labs under Projects Updates (www.elektormagazine.com/labs/200w-class-d-audio-power-amplifier-150511).

Elektor SDR Reloaded

Elektor 4/2016, p. 54 (150515), Elektor 5/2016, p. 60 (160048)

UPDATE. Dear Jan, I am very pleased to inform you that my software G8JCFSDR (Build 274) now fully supports the Elektor SDR shield. I have already been in contact with Burkhard Kainka, who has tested the “reloaded” version of G8JCFSDR.

The Elektor SDR shield support of G8JCFSDR provides a tuning resolution of 1 Hz without glitching, a convenient calibration tool (to ensure that your SDR shield is always precisely on frequency), and a built-in Arduino firmware upload function to simplify upgrading.

The G8JCFSDR software is available for download at www.g8jcfsdr.dyndns.org.

I would very much appreciate it if you would inform your readers about this new version.

Peter, GM8JCF

Dear Peter, thanks for your message. 5 and 9. We are more than happy to relay the word about this update.

Jan Buiting, PE1CSI

RETRONICS

Happy 40th Birthday, PET!

Breakthrough BASIC box: no soldering involved!

Visitors to the January 1977 Consumer Electronics Show (CES) witnessed a peculiar premiere in deep-frozen Chicago. Revealed here to an astonished audience was the first fully equipped, ready-to-use Personal Computer in the world: the Commodore Personal Electronic Transactor. In short: the PET 2001.

By **Karl-Ludwig Butte** (Germany)

Foto: Wikimedia Commons

The 'microcomputer' enthusiast scene had already begun to bubble about two years earlier, when *Popular Electronics* magazine caused a sensation by publishing an article about the Altair 8800 computer being offered in build-it-yourself kit form (**Figure 1**). Everywhere people tinkered diligently with homebrew computer creations, countless books appeared about 'how to build your own working computer' (**Figure 2**) and computer clubs, such as the Homebrew Computer Club [1], sprung up like mushrooms. But the time was now ripe for a computer that served up all you needed to participate in the new age of the computer: CPU, power supply, memory, monitor screen, keyboard and mass storage — all in a single unit! And the key feature: just seconds after powering up you were greeted by the Commodore BASIC Interpreter, ready to fulfill immediately your every BASIC command desire. No need to toggle binary switches, as on the Altair 8800, and no keying in of Hex values, as with the KIM-1 (Commodore's first teetering steps into this marketplace of the future). No, just clearly understandable (BASIC) words, such as `GOTO`, `PRINT` or `IF …THEN` for example, sufficed for programming the incredible capabilities of the PET 2001. This was truly revolutionary, even if we can hardly grasp its magnitude today, with our 8-core CPUs, gigabytes of memory, terabyte hard drives and graphical user interfaces.

Hardware

So what did you get then for $595 (or $2350 in today's value [2])? At the heart of the PET was a 6502 CPU that could access 4 KB of main memory. Upgraded with 8 KB main memory would cost you a surcharge of $200 [3]. For visual display a 9-inch monochrome monitor screen with white or green text was built in. The 'Chiclet' keyboard comprised 73 keys, which Commodore adopted from its desk calculator product line. This had the advantage that the keyboard did not occupy as much space as an actual typewriter keyboard would, leaving room alongside for a built-in cassette recorder using regular compact cassettes. This enabled you to save your data and programs for subsequent use. A second recorder could be installed externally using the so-called Datasette Port. Two additional interfaces assured the PET of a glorious future. First of these was the so-called Userport. This made accessible (among other things) the digital I/O connections of the 6502 processor, to which you could hook up hardware of your own. Most personal computers of this era, such as the Apple II, TRS-80 or the C64 for instance, had an expansion capability of this kind. Subsequently this option vanished from later generations of PCs until they enjoyed a renaissance in 2012, under the guise of 'GPIO Pins' with the Raspberry Pi. This user port accommodated not only hobby enthusiasts but also the newly founded enterprises everywhere, which developed additional hardware for use with the personal computer. The second interface, an IEEE 488 bus, was used by Commodore itself to connect peripheral devices such as printers and floppy diskette drives (**Figure 3**). The use of this bus system was noteworthy in several respects. The peripheral devices were connected to the PET in a daisy chain, one behind another, yet they could also be controlled by this bus in parallel. Commodore must have been unable to decide initially how many (and which) peripherals would be joining the PET. With its maximum of 15 devices, the bus offered more than enough options. Apart from this single IEEE 488 interface no further hardware was to be installed on the main board of the PET. This meant that any peripherals had to provide their own intelligence in the form of a complete microcontroller system, even though this made them more expensive to buy. Nevertheless Commodore saved valuable time developing the PET, which could be used to achieve earlier market entry.

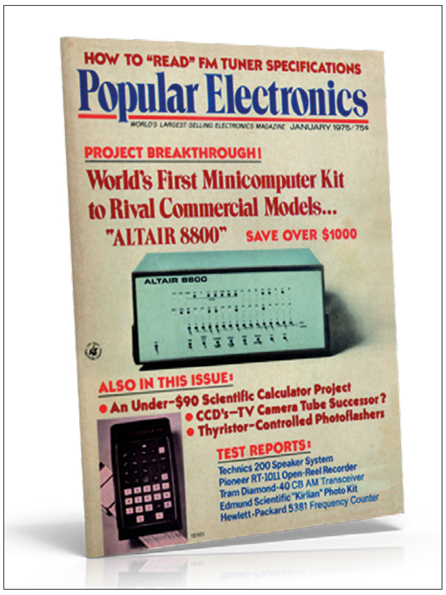

HOW TO "READ" FM TUNER SPECIFICATIONS
Popular Electronics
WORLD'S LARGEST-SELLING ELECTRONICS MAGAZINE JANUARY 1975/75¢
PROJECT BREAKTHROUGH!
World's First Minicomputer Kit to Rival Commercial Models...
"ALTAIR 8800" SAVE OVER $1000
ALTAIR 8800
ALSO IN THIS ISSUE:
An Under-$90 Scientific Calculator Project
CCD's—TV Camera Tube Successor?
Thyristor-Controlled Photoflashers
TEST REPORTS:
Technics 200 Speaker System
Pioneer RT-1011 Open-Reel Recorder
Tram Diamond-40 CB AM Transceiver
Edmund Scientific "Kirlian" Photo Kit
Hewlett-Packard 5381 Frequency Counter

Figure 1. Cover shot from *Popular Electronics* for January 1975 (Ziff Davis Publishing Company).

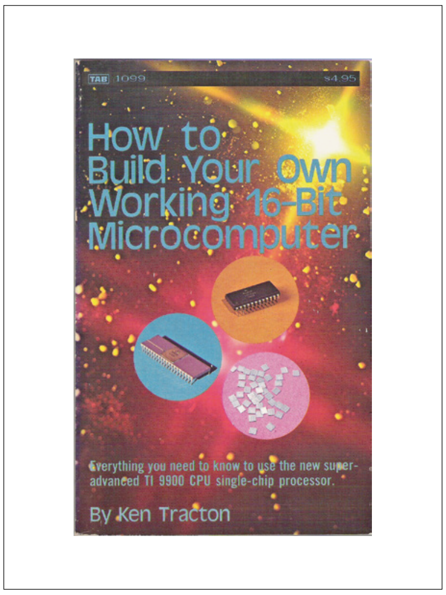

TAB 1099
$4.95
How to Build Your Own Working 16-Bit Microcomputer
Everything you need to know to use the new super-advanced TI 9900 CPU single-chip processor.
By Ken Tracton

Figure 2. One of the numerous home construction manuals.

Apple chose the opposite approach. Seven unallocated expansion slots were provided on the motherboard. Into these you could for instance plug a diskette controller, which constituted purely the electrical interface for the disk drive electronics. Control of the (maximum of two) floppy drives that could be connected was handled entirely by the Apple II CPU itself. Conversely the PET simply passed a command to the Floppy Station and could then dedicate itself back to other tasks after issuing a command such as 'Copy Diskette 1 to Diskette 2' (leaving the diskette station to execute the command automatically). The entire hardware was enclosed in a massive sheet-steel housing, which could be opened like the hood (bonnet) of a car. It even had a metal prop to keep it open (**Figure 4**). Evidently this robust construction had a bearing on how quite a few PETs have survived to this day too. They are offered occasionally on some well-known auction sites nowadays at prices between $200 and $800. Often these devices reach their new owners in a deplorable state. But if you are lucky what looks like heavily encrusted dirt (**Figure 5**) is only superficial and after an extensive clean-up session, they come up gleaming again in their original glory (**Figure 6**).

Software

The BASIC Interpreter on the PET combined both operating system and program development environment in one and was located in 14 KB of ROM (**Figure 7**). It was based on Microsoft BASIC and was enhanced with a so-called 'full-screen editor' [4]. In this manner you could go back and alter lines in a program visible on the monitor screen by moving the cursor to the relevant line of code and then editing it. After

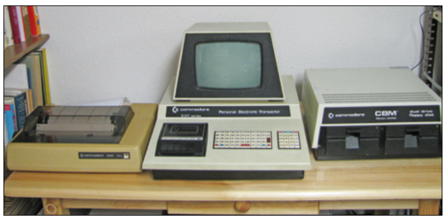


Figure 3. A fully developed set-up with PET 2001, printer and diskette station.

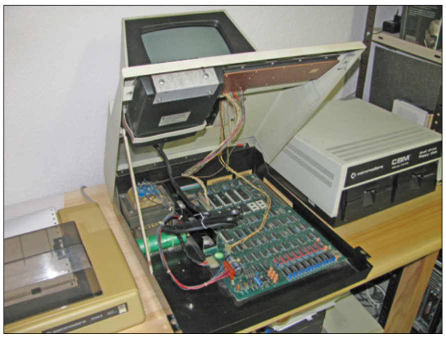


Figure 4. The case can be released by undoing just two screws.

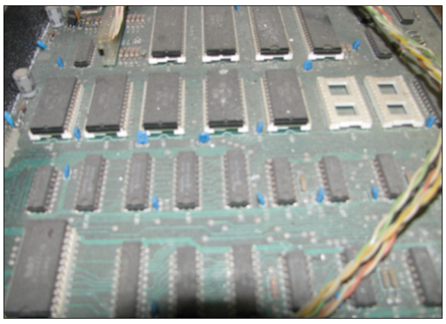


Figure 5. State of the main board when received by the author.

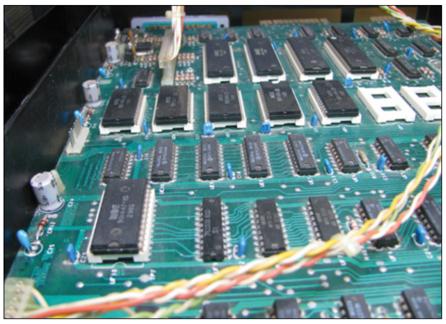
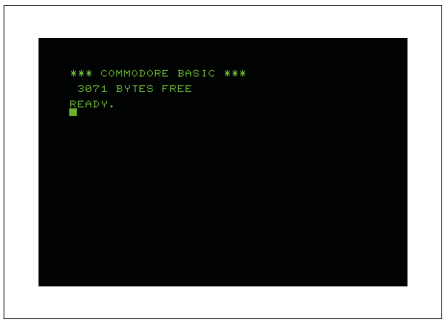


Figure 6. State of the main board following clean-up session.


```
*** COMMODORE BASIC ***
 3071 BYTES FREE
READY.
█
```
Figure 7. BASIC prompt on powering up (Tom Knox, *Wikimedia Commons*).

pressing the Return key the alterations would be taken on board. Seen against conditions prevailing at that time this was unbelievable luxury! In addition to the normal BASIC commands you had also the commands `PEEK` and `POKE`, with which you could read out individual Bytes from memory or correspondingly write them into memory.

Variable names were distinguished only by their first two characters. They could of course be longer but any subsequent characters were ignored. Consequently the BASIC Interpreter would, for example, read both the Variables `MONEY` and `MONTH` as the same Variable '`MO`'. Commodore BASIC recognized three different types of data: 40-bit floating point numbers, 16-bit integer numbers and character strings having up to 255 characters. These were signified with a special character at the end of the Variable name. Strings were suffixed with the `$` symbol, whilst Integer Variables were indicated by the symbol `%`. If a recognition symbol of this kind was missing, the Variable was treated as a floating point number. Variables could not be 'declared'. If a new Variable was recognized, the Interpreter reserved the number of Bytes corresponding to the data type in the memory. This had disastrous consequences if you made a typing error and you would puzzle over the totally inexplicable behavior of your program. Anyone who more than once typed in the Variable `INVOCE$` when `INVOICE$` was intended will know exactly how this felt. It often helped to go through the program with an uninvolved third party. According to their level of experience, they would spot keying errors far faster than you could yourself.

A BASIC Instruction for immediate execution was simple to enter. `PRINT 2*SQR(25)` <Return> would bring the result '10' onto the screen instantly. If a line number was written ahead of the Instruction, the PET delayed its execution until you entered the command `RUN`. After a maximum of two or three attempts at programming, the user would learn that it made sense to assign these line numbers in multiples of no less than ten. You see, if you had written a ten-line program and assigned the line numbers 1, 2, 3, etc. up to 10, this made it impossible to add an eleventh line between lines 4 and 5 without renumbering lines 5 to 10 manually to lines 6 to 11. At the same time you would need to remember to modify correspondingly the targets of the conditional (`IF`) and unconditional (`GOTO`) jump commands. In the beginning there was no `RENUMBER` command that could handle all this automatically. Yet the committed group of programmers (today we call them the 'community') plotted a remedy. Because it soon became clear that a BASIC program was not the best choice for renumbering another BASIC program from scratch, heavier weaponry had to be rolled out. The solution went by the name of Assembler — a program in machine language and accommodated in a protected range of memory should be in a position to achieve this miracle. Fortunately the PET came with the so-called *Machine Language Monitor* (initially only on cassette, subsequently in ROM). Now things really got interesting and every self-respecting PET programmer knew the 'secret command' for probing the depths of Assembler programming: `SYS 1024` <Return>. Machine language did indeed help in many situations where you could get no further using BASIC. Specialist magazines, books and printed newsletters from countless computer clubs printed pages of hex code listings. Using these you could implement essential things, such as the `RENUMBER` command for instance. You also discovered practical things, such as how to connect a barcode reader for capturing programs in this format.

Applications

Games were assuredly a major application area for the PET 2001. In this connection you must not envisage the kind of multicolor megapixel graphics that you can run on any old smartphone today. The PET didn't have even a proper graphics display on which you could control each pixel individually. No, it had just a range of graphic elements with lines, block or symbols to be fitted into the grid of letters and numbers that could be displayed on the screen. People called these 'block graphics' and were proud to be able to display bar graphs,

Web Links

[1] https://en.wikipedia.org/wiki/Homebrew_Computer_Club

[2] www.minneapolisfed.org

[3] www.computinghistory.org.uk/det/41502/Commodore-PET-2001-32/

[4] https://en.wikipedia.org/wiki/Commodore_BASIC

www.pcmuseum.ca/details.asp?id=194

http://home.total.net/hrothgar/museum/PET2001/

www.old-computers.com/museum/computer.asp?c=191

www.kuto.de/cbmmuseum/cbm_pet.html (in German) screen borders and even some games. Mathematical applications ranged from simple calculations through repayment plans right up to sophisticated linear equation-solving programs. Useful here were mathematical commands in the BASIC, such as `LOG` (logarithm) and `SQR` (square root).

With the arrival of printers and diskette stations, commercial applications also became possible, such as order processing, accounting and warehouse management. Small businesses in particular dared to take the plunge, because a Commodore system (with a computer, printer and diskette station) was still far cheaper than entry-level models from the established major computer manufacturers. A real killer app, as we would say today, came onto the market in 1979, in the form of the first spreadsheet program, *VisiCalc*. Originally written for the Apple II, the software was ported to the PET shortly afterwards. Also the first text editor (word processing) programs brought the PET into many an office.

Many hobby programmers gained their first experience hooking up additional hardware to the user port. You could also find ever more instructionals on how to entice audio tones out of the PET. All kinds of other minor measurement and control tasks could be solved with the PET too.

Verdict

The PET not only made computer history but also economic history. It kickstarted a transformation that introduced IT to the everyday workplace in organizations and initiated the emergence of an entirely new business market. Entire professions disappeared, such as typesetters in printing works, whilst new ones were established in record time, such as commercial PC programmers and IT consultants.

Probably only a few people perceived the full extent of the changes to come when, at the end of the 1970s, they took their first steps into the new world of computers with the PET. ⏮

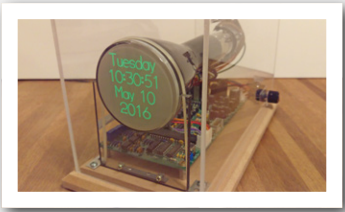
(160353)

Delayed Trigger

Compiled by Jan Buiting

Since its inception in 2004 the articles published in Elektor's Retronics section have drawn substantial response from readers and not limited to writing. Below are a few instances of donations and other contributions received over the years, along with the original article that was the 'trigger source' for readers to respond. For reasons of privacy, the names of the contributors are withheld unless specifically permitted to print here.

Object 04: Elektor MOS Clock 5314



The MOS Clock article triggered a wave of response! Alan Woodman built many digital clocks in the early years of their introduction, and now enjoys watching his 'scope clock pictured here. Reginald Neale put the larger MM5316 on perfboard. Bernd Rinke scanned the original 1974 article and reworked the PCB layouts; they are free for all to download from www.elektor.com/160102. Philippe Decock sent me a hilarious brain dump of his cherished reminiscences and also pointed out a Missing Link!
Status: we are gratified and amused.
Trigger 04: MOS clock 5314 (1974), Elektor 5/2016.

Object 05: Roberts Radio type 49041

"It's a small world after all" — both my dentist and my MD read Elektor and always plague me with dictums on the content especially on physics and formulae. Along with his prescription and compliments my MD Gaspard Knops gave me a Roberts 1950s battery-tube radio, commenting that he had "digested" my article on the Philips Colette radio, and "enjoyed" Ronald Dekkers' story on a Pye P87BQ Radio. One reason for the popularity of these battery tubes with many Elektor readers and authors might be that they can operate from relatively low anode voltages. I say these DF's and DL's are simply charming devices to work with.
Status: to be restored.
Trigger 05: Pye P87BQ Radio Find & Restore, Elektor May 2015.

Object 06: Electrofact pH Meter

Many electronicists career to medical equipment design, and countless Elektor Labs trainees have found employment at nearby Pie Medical. Medical studies involve chemistry and physics and I reckon this big pH meter from the tube era is best operated in a safe environment like an analyst's lab! Electrofact apparently was a Dutch firm. The instrument was donated to me complete with glassware and some sensors. No documentation has survived and apart from the tubes inside I'm in the dark about this one. Electrically, it works.
Status: to be explored & restored.
Trigger 06: Radiometer PHM22 / PHA928a Blood pH / O2 / CO2 Analyzer System, Elektor 9/2013.

Feel free to post-trigger and be listed here:
Email: editor@elektor.com; **Subject:** trigger [nn].

(almost) everything you wanted to know about...

Smart Buildings *vs.* Smart Homes

Smart homes are all the hype these days.
We asked Emmanuel Francois to elaborate on his expertise: smart buildings.

Q *Is there a big difference between home automation and smart buildings?*

A I would say there are two major differences: a smart home has less technical equipment than a smart building, and in a smart building efficiency is much more important. But another difference is that in smart buildings the focus is more on local IT solutions. The BMS (Building Management System) is largely 'analog' and not necessarily interactive or remotely controllable. A smart home is more likely to be fully 'digital' and remotely controllable via the Internet. The recently released "Home" application (available on most Apple and Android devices) lets you control connected home systems and devices and is kind of 'proof' of this difference. What's happening in smart homes now will follow in smart buildings. Smart home technology is kind of a precursor of what is coming to smart buildings.

Q *Are there (software) protocols similar to the ones used in home automation?*

A First, "smart" suggests the incorporation of IT solutions, both in smart buildings and home automation. Therefore, in both situations you will need an IP backbone. More and more I am seeing TCP/IP as the protocol of choice, because of its compatibility with the Internet. This also has to do with the demands for smart buildings: they need to be both flexible and scalable. The most logical solution is to go wireless. For wireless solutions there exist standardized protocols like Wi-Fi and Bluetooth (LE), so these are used. For example, most of the LED lighting systems use Bluetooth LE nowadays. These systems are mostly connected to an IP backbone. For local networks ZigBee, EnOcean [1] and Thread [2] are the most popular ones. KNX is a well-known protocol, but it is not really used in wireless applications. There are other protocols of course, but usually the protocol of choice is an open protocol. You will hardly see smart homes and smart building using a proprietary protocol.

Q *What are the benefits of a smart building system?*

A Top ranking: efficiency! Energy efficiency as well as space efficiency. For example, nowadays in France around half of the available desks in an office building are unused because of teleworking. Just saving 10% on the space needed for workplaces in a 10,000-m2 office can save as much as € 1,000,000 per year, so the use of office space needs to become flexible. Secondly, the productivity of the employees can benefit through a well-adapted working environment, ensuring comfort, clean air and good lighting conditions. And thirdly there is asset management. With a smart BMS you can optimize the building control.

Q *Which project do you think is the best example of what is possible with smart building technology?*

A My personal favorite is The Edge in Amsterdam [3]. It is very efficient, both energy and spacewise. Initially the building was planned to be 100,000 m^2. Thanks to smart solutions for space management it ended up being only 60,000 m^2, saving almost half of the initial space and reducing the costs significantly. If you look at it differently: one could say that the surrounding buildings have lost 40% of their value due to the efficient design of The Edge.
If we're talking smart homes, the French construction company Bouygues is building around 10,000 fully connected smart homes using "Flexom" [4] per year now. They are able to do

this with hardly any extra costs involved by using wireless connections and the EnOcean protocol. This saves a lot of money on the installing part of the smart systems. On top of that, they used NFC for the commissioning of the systems, resulting in a mere 20 minutes of time needed per home to prepare all the integrated smart solutions.

Q *When do you think this technology will reach its peak?*

A The key to the answer is the confidence of the user in the technology and in the solutions. That's why we have worked on a standard for smart buildings, called Ready2Services, which can be integrated in standards like HQE [5] and BREEAM [6]. Since we are working in a rapidly changing IT environment, this standard assures investors the SBM system will stay up to date and is future proof.

Q *Integration between smart homes and smart cars, when will that happen?*

A I think in 15 years time there will hardly be any combustion engine powered car left on the road. I'm sure by 2025 countries like Sweden will even decide to prohibit combustion engines. So as a result, almost every vehicle will be fully electric. For sure the battery, and thus the whole electrical car system, will be linked to the energy system of your smart home.

Q *What is determining the research at the moment?*

A Well, we spoke about smart cars en smart homes. The most important thing in both is identical: the brains. Artificial intelligence is leading the innovation, because the user demands that smart cars and smart homes adapt themselves to the user. This is a big issue for all the players on the field. They all need to be leading in artificial intelligence or they will become backseat players.

(160259)

Who is Emmanuel François?

Emmanuel has always worked with smart buildings. In 2014, at 55, he became Chairman of the Smart Building Alliance for Smart Cities (SBA), which he founded together with Serge Le Men of Newron System in 2012. With 170 members the SBA guides all involved parties with the transition of smart buildings and smart cities into the digital world. SBA also produced Ready2Services and Ready2Grid, two prerequisites for smart buildings and smart cities.
He is also sales manager for EnOcean GmbH, responsible for Western Europe. He has resided in the electrotechnical universe as a general manager or sales manager successively at Wieland Electric France, CAPRI (Cooper Group), Soulé (ABB Group) and Merlin Gerin (Schneider Electric).

Weblinks

[1] https://en.wikipedia.org/wiki/EnOcean

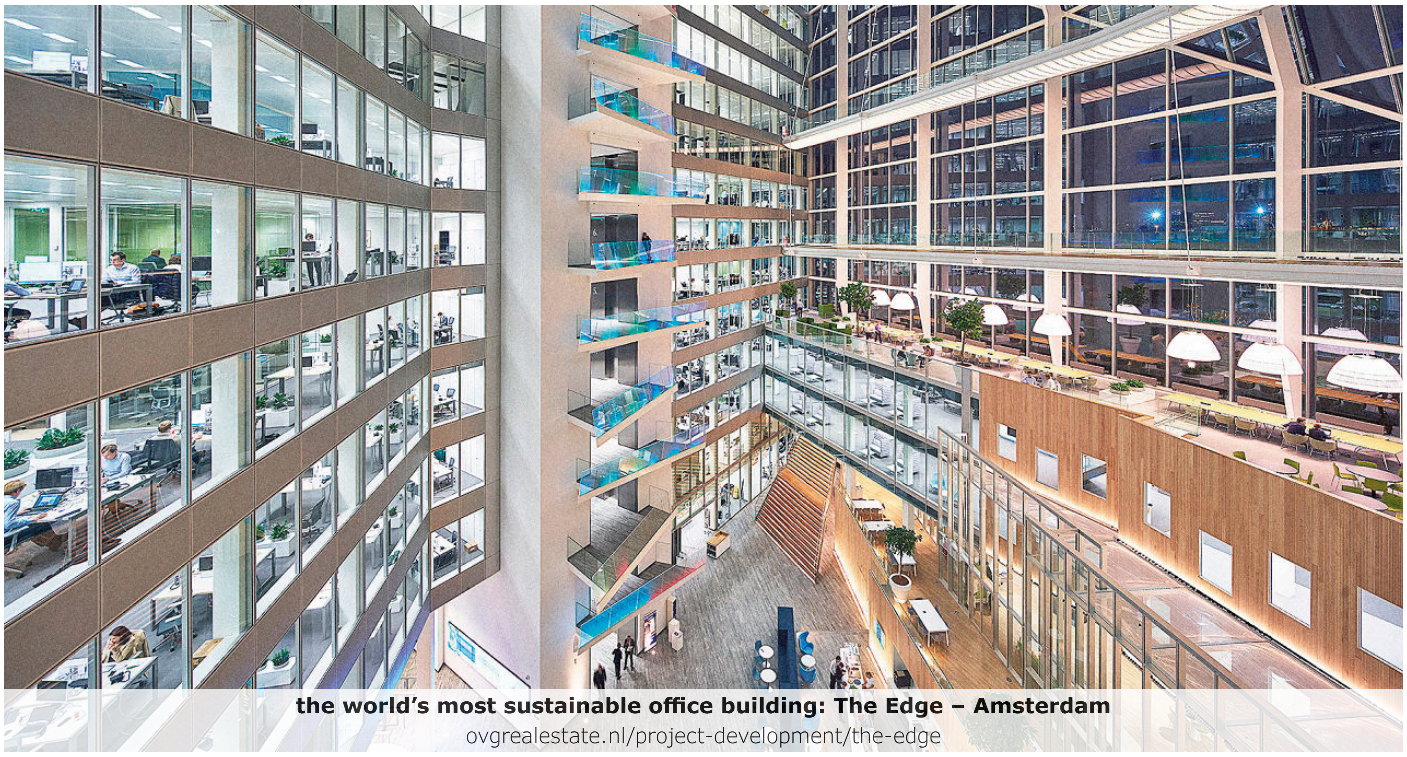
[2] https://en.wikipedia.org/wiki/Thread_(network_protocol)

[3] http://ovgrealestate.com

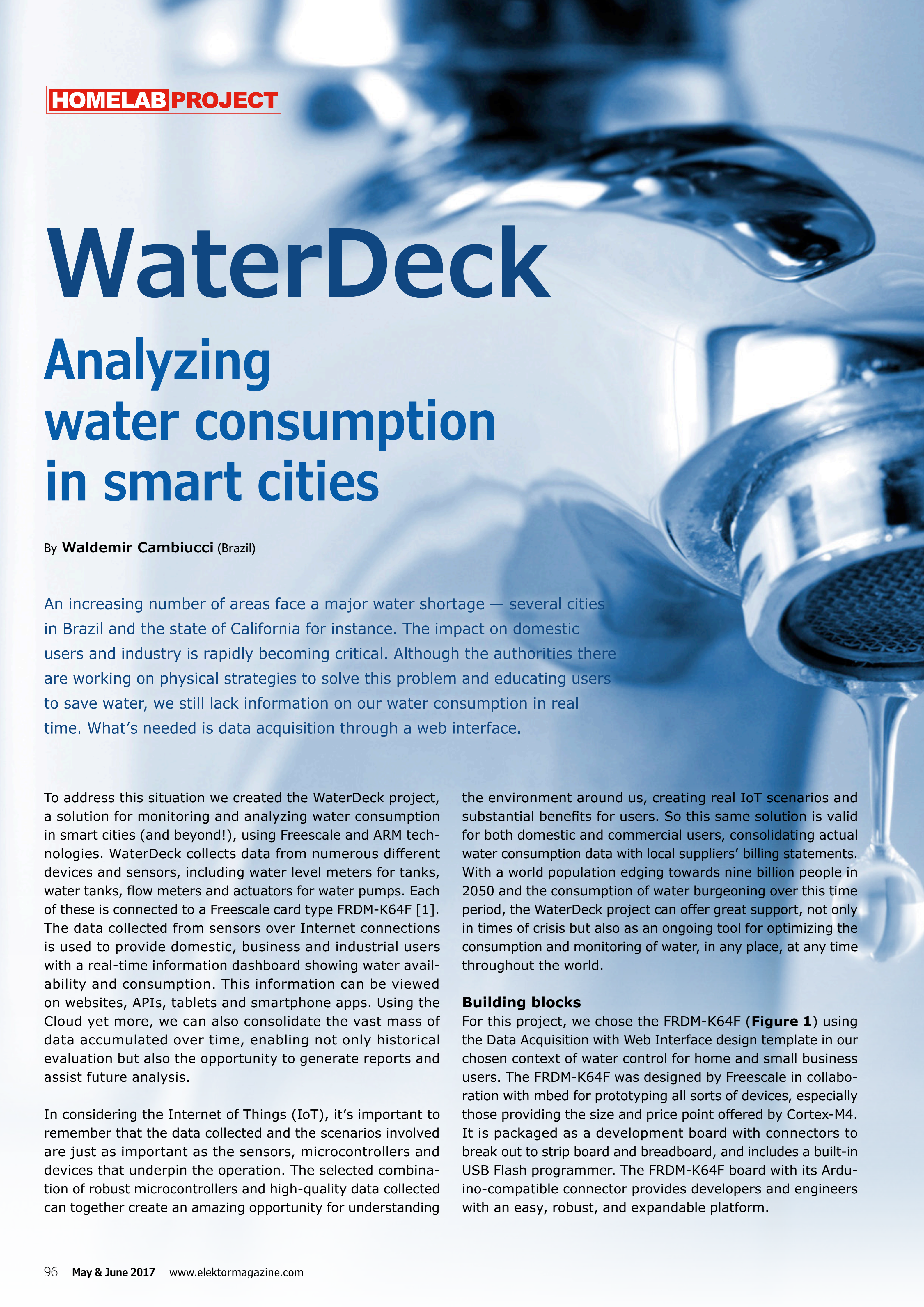
[4] https://flexom.bouygues-immobilier.com (French website)

[5] https://en.wikipedia.org/wiki/Haute_Qualité_Environnementale

[6] www.breeam.com



the world's most sustainable office building: The Edge – Amsterdam
ovgrealestate.nl/project-development/the-edge



HOMELAB PROJECT

WaterDeck

Analyzing water consumption in smart cities

By **Waldemir Cambiucci** (Brazil)

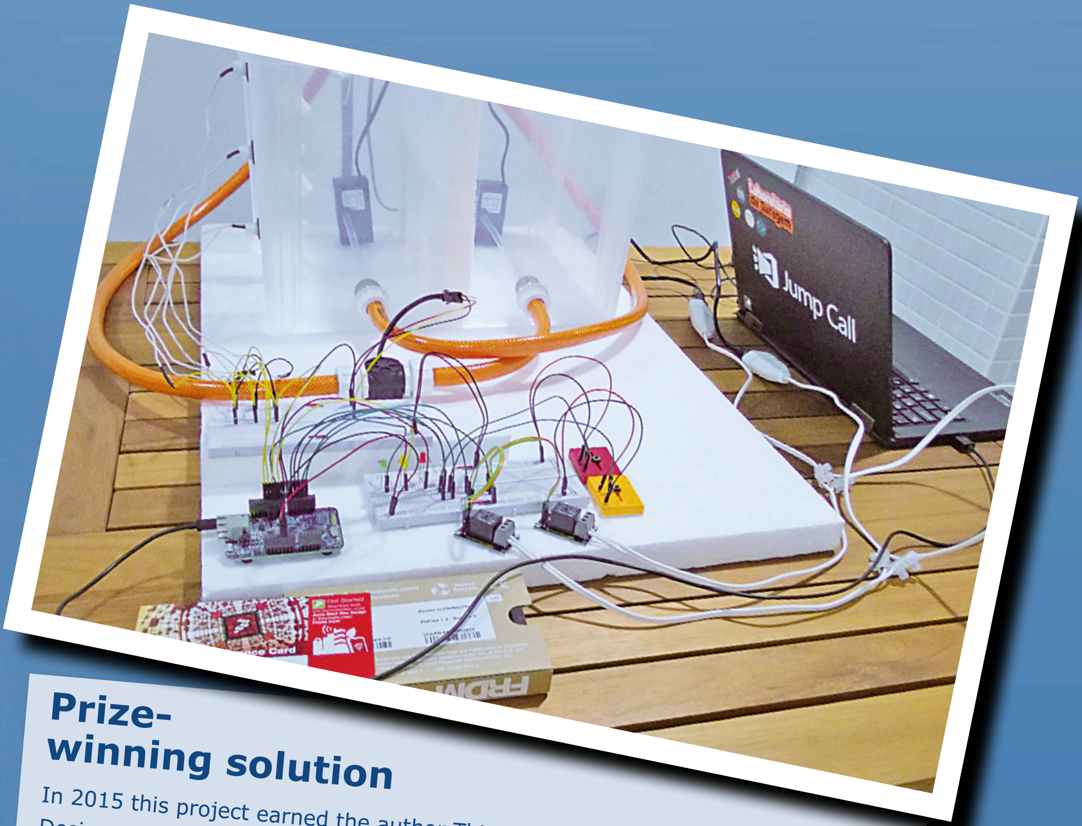
An increasing number of areas face a major water shortage — several cities in Brazil and the state of California for instance. The impact on domestic users and industry is rapidly becoming critical. Although the authorities there are working on physical strategies to solve this problem and educating users to save water, we still lack information on our water consumption in real time. What's needed is data acquisition through a web interface.

To address this situation we created the WaterDeck project, a solution for monitoring and analyzing water consumption in smart cities (and beyond!), using Freescale and ARM technologies. WaterDeck collects data from numerous different devices and sensors, including water level meters for tanks, water tanks, flow meters and actuators for water pumps. Each of these is connected to a Freescale card type FRDM-K64F [1]. The data collected from sensors over Internet connections is used to provide domestic, business and industrial users with a real-time information dashboard showing water availability and consumption. This information can be viewed on websites, APIs, tablets and smartphone apps. Using the Cloud yet more, we can also consolidate the vast mass of data accumulated over time, enabling not only historical evaluation but also the opportunity to generate reports and assist future analysis.

In considering the Internet of Things (IoT), it's important to remember that the data collected and the scenarios involved are just as important as the sensors, microcontrollers and devices that underpin the operation. The selected combination of robust microcontrollers and high-quality data collected can together create an amazing opportunity for understanding the environment around us, creating real IoT scenarios and substantial benefits for users. So this same solution is valid for both domestic and commercial users, consolidating actual water consumption data with local suppliers' billing statements. With a world population edging towards nine billion people in 2050 and the consumption of water burgeoning over this time period, the WaterDeck project can offer great support, not only in times of crisis but also as an ongoing tool for optimizing the consumption and monitoring of water, in any place, at any time throughout the world.

Building blocks

For this project, we chose the FRDM-K64F (**Figure 1**) using the Data Acquisition with Web Interface design template in our chosen context of water control for home and small business users. The FRDM-K64F was designed by Freescale in collaboration with mbed for prototyping all sorts of devices, especially those providing the size and price point offered by Cortex-M4. It is packaged as a development board with connectors to break out to strip board and breadboard, and includes a built-in USB Flash programmer. The FRDM-K64F board with its Arduino-compatible connector provides developers and engineers with an easy, robust, and expandable platform.



Prize-winning solution

In 2015 this project earned the author Third Prize in the ARM Keil Microcontroller Design Contest to create a smart domestic or industrial device based on an ARM Cortex-M processor. The competition attracted more than 700 entries. Waldemir sums up his experience:

I definitely consider that the ARM Microcontroller Design Contest organized in cooperation with Elektor was a wonderful opportunity to improve my personal skills in IoT, learning many aspects from using such a new card as the Freescale FRDM-K64F. It was also great way of connecting with new friends and enthusiasts around the world via the ARM Community. Finally it was an honor to be selected as one of the winners of this contest and I'm already awaiting the next ARM Keil contest.

During the 2015 ARM Microcontroller Design Contest (see inset **Prize-winning solution**) we were able to select one out of four different templates available. We chose the template for Data Acquisition with Web Interface that sampled a middleware network component to display incoming sensor data from an accelerometer and magnetometer on a web page.Middleware is a layer of software that acts as an interface between two dissimilar pieces of computer code to make them work together, such as between an application program and its operating environment. Using CGI scripting and JavaScript, users can change the recording interval of the incoming data.
The goal (**Figure 2**) was to collect the multiple pieces of information from the sensors, water level gauges, pumps, flow meters, etc., and create 'push' notifications to the end-user in a web interface. In this way it was possible to display the dashboard on any device, such as tablets, smartphones or desktops, from a dataset hosted in the Cloud.

WaterDeck in action

For this project, we created a complete simulation environment, using two tanks to simulate water consumption. The first container was a regular domestic water tank that was fitted with three water level sensors from ICOS [2]. Using these sensors, it was possible to connect to the Freescale FRDM-K64F via a normal digital Pin, reading the signal as On or Off.

In the same way, we used a water flow sensor that we installed in the main supply line, ahead of (before) the main pump. So, whenever water flows into the main tank, it's possible to read

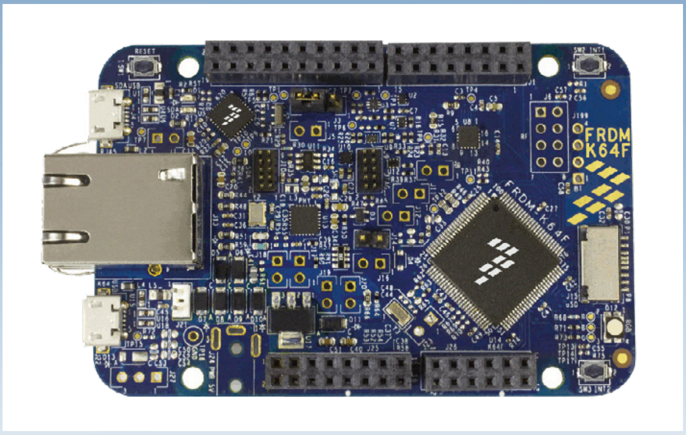


Figure 1. Freescale FRDM-K64F board.

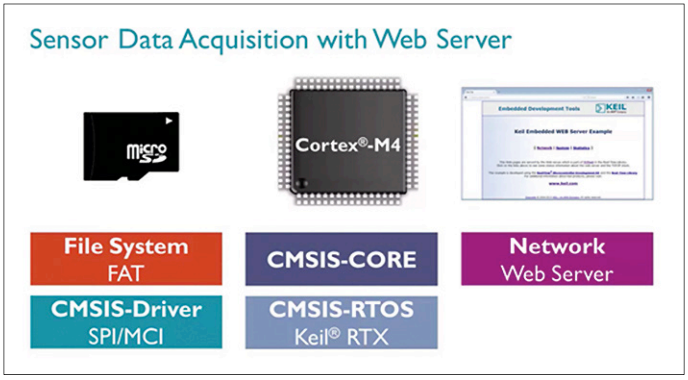


Figure 2. Sensor data acquisition with web server.

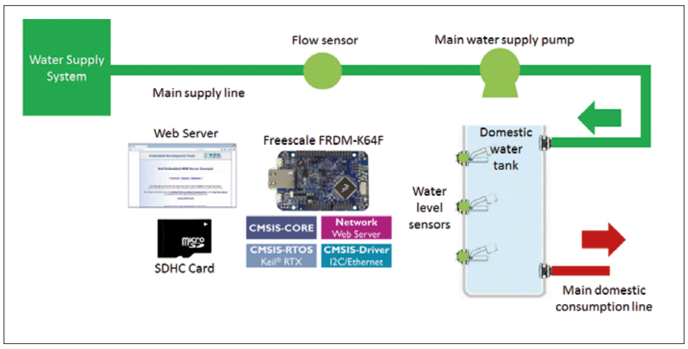


Figure 3. Using the Freescale FRDM-K64F board to control and monitor water consumption for domestic solutions, presenting information via the Internet.

how many gallons per hour or liters per minute are entering the domestic water tank.
Finally we have another water pump simply to simulate consumption from the domestic water tank. When this pump is activated, water is drawn from the main tank and returned to the external tank.
The Freescale FRDM-K64F is the centerpiece of this simulation, controlling the numerous digital inputs and outputs of the system. A great feature of this board is the ability to export all data collected to the web, enabling a number of potential users to access this data.

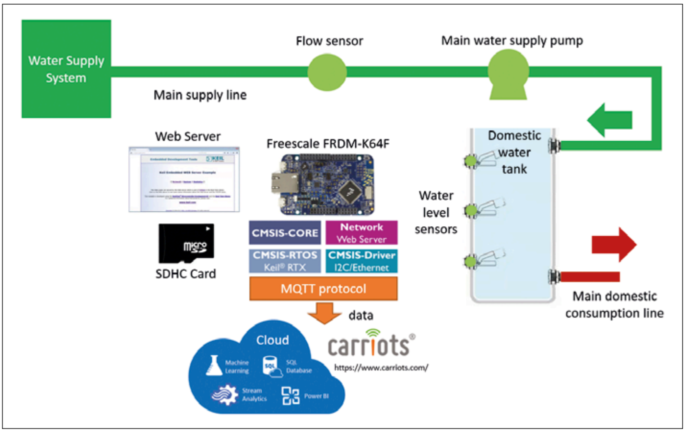


Figure 4. Diagram for the main circuit, including the adapter for the MQTT protocol, connecting Freescale with different dashboards or IoT platforms hosted in the cloud.

In the same way, we use an SDHC Card to save data during the simulation to archive the historical behavior of the many sensors and components in the system. We can use this data collected during the simulation to feed websites or other systems in the web.

Connecting to the Cloud

Still using the Data Acquisition with Web Interface design template, it is possible to visualize this project as a pilot for connecting the Freescale platform with different IoT dashboards in the Cloud, such as Windows Azure and Amazon Web Services, or with SaaS (software as a service) solutions such as the Carriots IoT platform [3].
To do this, we need to use an adapter or a client for MQTT Protocol, allowing the system to connect with different vendors and send messages to be consolidated. MQTT stands for Message Queuing Telemetry Transport and supports many possible scenarios for messaging in IoT projects. We are currently examining the Mosquitto solution to connect with Freescale in our project for the future. Mosquitto is an open source (BSD-licensed) message broker that implements the MQ Telemetry Transport protocol versions 3.1 and 3.1.1. Further information can be found at [4]. The diagram in **Figure 3** presents an overview of this context.

Conclusions

As this article shows, our project was designed to create a complete display of data and monitoring of water for domestic consumption. It was based on water tanks, water level meters for tanks, flow meters and actuators for water pumps in widespread use. It used a standard Internet connection to provide a real-time dashboard display about the availability of water in the user's home, viewable on websites, APIs, tablets and smartphone apps handling this information.

During the project it was possible to evaluate and to exploit many important features of the Freescale FRDM-K64F, which proved to be a very versatile card for IoT applications. The exercise was also a great opportunity to experience the Keil uVision5 development environment with which the project was configured and tested.
Important features were used in this project, were:

- log file with SDHC card;
- interrupts for acquiring data from the water flow sensor;
- pins for input and output valves, also LEDs and sensors of various types;
- relay to control external 220 V circuits, such as water pumps;
- pushbuttons enabling the user to provide input manually.

(150502)

Web Links

[1] www.nxp.com/products/software-and-tools/hardware-development-tools/freedom-development-boards/freedom-development-platform-for-kinetis-k64-k63-and-k24-mcus:FRDM-K64F

[2] www.icos.net

[3] www.carriots.com

[4] http://mosquitto.org

sigfox bringing objects to life

With the aim of bringing millions of objects to life, Sigfox has set up a network capable of connecting billions of objects to the Internet. This network is currently present in 29 countries and thus accessible to around an estimated 470 million people. Companies wishing to develop services and products in the field of the Internet of Things (IoT) now have a wide range of solutions at their disposal, thanks to Sigfox's increasing number of partners. Let's set off to explore the world of Sigfox.

By **Mariline Thiebaut-Brodier** (Editor-in-Chief, Elektor France)

In the field

This French company was set up in 2010 by Ludovic Le Moan and Christophe Fourtet and is based in Labège, near Toulouse, in the heart of 'IoT Valley'. It has set up a narrow-band cellular network using the non-licensed frequency bands dedicated to communication between machines (M2M) and IoT.

Currently present in 29 countries (mostly in Europe), the Sigfox network should be extended to around sixty more countries by 2018 (see [1]). Sigfox is installing stations, but is also being deployed via alliances with telecoms operators like SFR (France) or Albertis Telecom (Spain). These operators' customers enjoy the benefit of the Sigfox connectivity in addition to the existing services (4G, Wi-Fi, etc.).

The company has also entered the world of smart metering for remote reading and consumption monitoring (water, electricity) as well as distribution network surveillance (detecting water leaks, electrical load curve). Sigfox has formed alliances with cloud service providers like the Microsoft Azure cloud. The data collected by Sigfox objects is transmitted to the Azure platform for processing.

Very diverse sectors are now looking for connectivity: smart cities, logistics, agriculture, security (domestic, corporate), vehicles, environment, and so on. Remote tracking (GPS tracking) for objects, animals, and people is a fast-expanding sector. The Sensolus company is offering the Chickenrun system which allows time-of-flight to be calculated for poultry. The French HidnSeek company (see **inset**) sells various self-contained tags that transmit the position of the connected subject or object, along with temperature and atmospheric pressure.

Technology

Sigfox has deployed a narrow-band (100 bit/sec) network that transmits short messages over long distances. The distance record for production models is 1,024 km (2^{10}) between an object and a 'station', but in reality it varies between 2 and 130 km, depending on the topology of the terrain. The connected objects then consume very little power for the radio transmission, which is bidirectional and uses an ultra-narrow band (UNB). The transmitter modulates the transmitted carrier using binary phase-shift keying (BPSK). The receiver listens to a very narrow band of the spectrum in order to detect the messages. Then the valid received messages are sent to the Sigfox cloud or to the platform chosen by the object's owner. The network is bidirectional, but only when the object requests it. It is important to appreciate that in the Sigfox protocol, the network serves the object, not the other way around. Thus the object itself requests an "update" from the network, which responds if necessary. The rest of the time it remains "deaf". This technology has been designed and optimized for battery usage; certain objects already in production are intended to be able run for up to 10 years on ordinary batteries. In point of fact, it's very easy to calculate the current consumption of a Sigfox object, as it is constant, unlike what happens with other technologies. This makes implementation a great deal easier. Obviously, the battery life is connected to the number of messages that need to be sent per day (max. 140 messages per object per day). Each message contains a maximum of 12 bytes of useful data.

As a company, Sigfox sets itself apart in both technological and commercial terms from the other "LPWAN" initiatives. Sigfox

Table 1. Sigfox coverage around the world.

Radio communication zone	Region
RCZ1	Europe (868 MHz, ETSI 300-220), Oman, South Africa
RCZ2	United States (902 MHz, FCC part 15), Mexico, Brazil
RCZ3	Japan (923 MHz, ARIB STD-T108)
RCZ4	Australia, New Zealand, Singapore, Taiwan, Hong Kong, Colombia, Argentina (920 MHz, ANATEL 506, AS/NZS 4268, IDA TS SRD)

operates with a very open approach for its protocol, since all semiconductor manufacturers can incorporate the Sigfox protocol into their products free of charge. Unlike the competing technologies, which oblige them to pay license fees on every chip produced, leading to a *de facto* monopoly, Sigfox earns income only via the subscriptions to its service. This approach ought to encourage competition between module manufacturers and exert downward pressure on the price of objects. The cheapest modules for entering the Sigfox world are on offer at around €2.
The engineering structure of the network too is very different from the other technologies, which is contributing greatly to its rapid success worldwide. It seems to require a much smaller number of stations than its competitors to cover an equivalent zone (thanks to its extended range) and to have a much higher capacity for the number of objects connected.

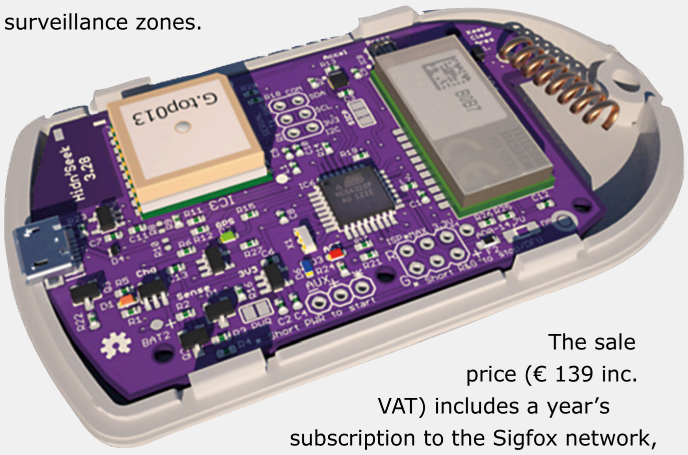
Practical implementation

For sending messages on the Sigfox network, you need Sigfox-compatible equipment — a module or a development kit. First of all, you need to choose the zone you will be working in. **Table 1** details these various zones, established primarily according to the transmission and radio spectrum standards to be abided by.
Then you need a transceiver that speaks Sigfox, or a transmission module containing one, along with a microcontroller. **Table 2** gives a number of references for circuits of this type. The modules are driven either directly using AT commands (low-level communication) over a serial interface (UART) or via an API interface (libraries).
If you're a seasoned Arduino or Raspberry Pi user, you won't have any trouble finding a Sigfox expansion board for your favorite board. Everyone else can turn to a self-contained development board. In the **insets** below we have gathered together a selection of boards and kits currently available. Most of these have obtained the "Sigfox Ready" seal of approval after passing a dedicated certification process managed by Sigfox. Type certification costs €2K5 for an object. As part of this process, the power transmitted by the object tested is measured, in order to classify it according to a scale based on the ETSI or FCC standards (see **Table 3**). Class 0u ensures good coverage for the object.

Let's get back to those kits. The pleasant surprise is that their price often includes a year's access to the Sigfox network, so you're all set to start experimenting as soon as you open the box. Network access alone costs between €1 and €20 a year. Sigfox is making life easy for startups by offering network subscriptions, along with free tools, to those who are seeking to connect one of their products. There is even a dedicated "startup" program in order to speed up seedling projects.
If you go for a development kit or a system-on-chip (SoC) without subscribing to your chosen product manufacturer's own platform, you'll have to register your object on a platform so you can follow it — see the list in [2]. Some of these are reserved for specific applications like waste or bee tracking.

HidnSeek – ready-to-use Arduino IDE-compatible board

The HidnSeek company sells the HIDNSEEK ST1-A/B tracker [4], an autonomous rechargeable standalone (no SIM card) GPS tracker. A mobile application in conjunction with a website make it possible follow one or more trackers and define alerts and virtual surveillance zones.



The sale price (€ 139 inc. VAT) includes a year's subscription to the Sigfox network, renewable annually (€19.90).

The most interesting thing is that the APIs and the firmware for this object are open. So it can be turned into a development board that can be adapted for your own use by adding sensors. The manufacturer makes all the tools needed to reprogram the object (using the Arduino IDE and a USB cable) available on GitHub ([5] and [6]). The board carries an ATmega328P microcontroller @ 12 MHz, a Sigfox TD1207 bridge, and a PA6H GPS module from GlobalTop.

Specifications

- GPS localization
- 3-axis movement detector
- Temperature and pressure sensors
- Firmware updating via USB interface
- Geofencing using Apple iOS and Android apps.
- General-purpose I/O (GPIO) free for extensions (I²C, serial, ISP, analog/digital I/O)
- Rechargeable, monitored 500 mAh LiPo battery (battery life of several months)

Others are general in nature and can be used by anyone, like Microsoft IoT, Amazon AWS, or Thethings.io.
The following information allows each Sigfox object to be identified:

- A unique identifier, SIGFOX ID (read-only device unique identifier). This ID normally appears on a label stuck to the object. It cannot be modified and is vital for registering an object on the network (be careful not to lose it!). E.g.: 16B87.
- A PAC code (Porting Authorization Code). This code is also essential for registering an object, but it changes if the object changes owner. E.g.: CF14xxxx-xxx.
- A hidden security key, AES128 encrypted.

The list of products suggested here is not exhaustive. It simply shows that the product offering for landing on planet Sigfox is varied. You are spoilt for choice! What's more, Sigfox has set up a dedicated website for developers (see [3]), offering tutorials, along with links to various resources.
To sum up, the modules available make it easy to create a connected object. Access to the Sigfox network is simple and not expensive, compared to Lora, for example. So you have all the tools you need to produce your first connected object. Will it be for keeping track of your dog who's always running away; or for collecting environmental data in your orchards? We're sure you'll find some original uses for the Sigfox network. And don't forget to share your experiences with us.

(160167)

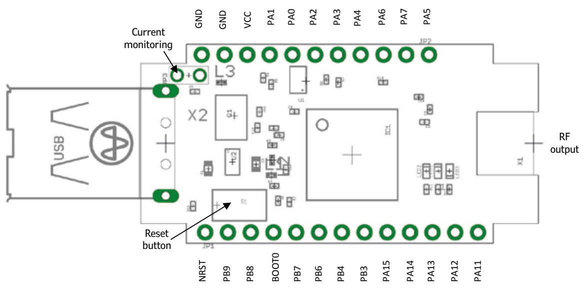
Table 2. ICs for Sigfox communication, used in various shields, development kits, and expansion boards.

Manufacturer	Reference	Type	Practical implementation
Atmel	ATA8520D-GHQW	transceiver + 8-bit AVR microcontroller	
	ATA8520E-GHQW	transceiver + 8-bit AVR microcontroller	evaluation kit: ATA8520-EK4-E (RCZ1) / ATA8520-EK1-F (RCZ2); expansion board for Xplaine Pro: ATA8520-EK6-E (RCZ1) / ATA8520-EK3-F (RCZ2)
	ATA8520-GHQW	transmitter + 8-bit AVR microcontroller	
OnSemiconductor	AX-SFEU / AX-SFEU-API	transceiver	development kit: DVK−SFEU−[API]−1−GEVK
	AX-SFUS / AX-SFUS-API (RCZ2)	transceiver	
Radiocrafts	RC1682-SIG	radio module (controller + Sigfox stack)	demonstration kit: RC1682-SIG-DK
STMicroelectronics	S2-LP	transceiver	development kit: STEVAL-FKI868V1 or STEVAL-FKI915V1 (NUCLEO-L152RE board + S2-LP)
TD Next	TD1204	Sigfox transmitter + GPS receiver	evaluation board: TD1204 EVB
	TD1205P	Sigfox transmitter + GPS receiver + antenna + accelerometer	
	TD1207R/08R	bridge (transmitter + ARM Cortex-M3 microcontroller)	evaluation board: LGA25 EVB
	TD1508 (RCZ2)	bridge (transmitter + ARM Cortex-M3 microcontroller)	evaluation board: LGA25 EVB
Telit	LE51-868 S	transceiver	evaluation kit: LE51-868S EVK
Texas Instruments	CC1120 / CC1125 / CC1190	transceiver	development kit: MSP430F5529 LaunchPad or CC1120 CC1190 BoosterPack or CC1125 BoosterPack
Wisol	WSSFM10R1	transceiver	expansion board: BRKWS01 from SNOC

Table 3. Uplink (from object to network) classes depending on radiated power.

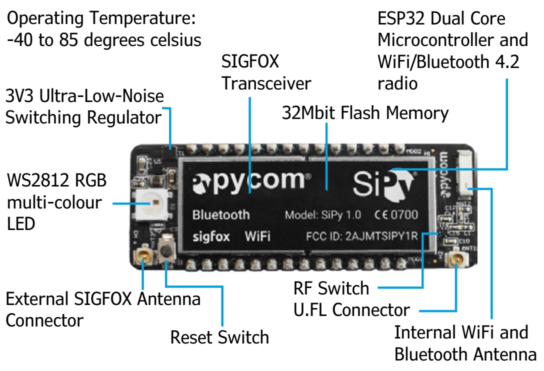
	Class 0u	Class 1u	Class 2u	Class 3u
ETSI	14 dBm ±2 dB	12 dBm > P > 7 dBm	7 dBm > P > 0 dBm	< 0 dBm
FCC	22 dBm ±2 dB	20 dBm > P > 15 dBm	15 dBm > P > 5 dBm	< 5 dBm

Nemeus – USB stick



Nemeus are offering the MM002-LS-EU [7], a bi-mode (LoRa/Sigfox) modem that includes an STM32L151X microcontroller and an RF SX1272 transceiver. The whole thing is run by the Contiki operating system. The JTAG connector allows you to load your code and the I²C, SPI, UART, GPIO, ADC, DAC ports allow dialog with sensors.
To get experimenting using the Sigfox network faster, you can use the ready to use version of the MM002-LS-EU: the MK002-LS-EU USB stick [8]. All you have to do is connect it to a computer (Windows or Linux) to send it AT commands. A JAVA app is provided for testing. It is also possible to power the stick directly (Vcc pin) and to access the UART (pins PA2 and PA3).
At the time of writing, Nemeus was announcing the launch of the IoT Smart Sensor, a board for experimenting with wireless links (Sigfox, LoRa, Bluetooth, GPS/GNSS, etc.). This Arduino IDE-compatible board includes an accelerometer, temperature sensor, and pressure sensor, as well as a user-programmable RGB LED.

PyCom – programming in Python


Operating Temperature: -40 to 85 degrees celsius
ESP32 Dual Core Microcontroller and WiFi/Bluetooth 4.2 radio
SIGFOX Transceiver
3V3 Ultra-Low-Noise Switching Regulator
32Mbit Flash Memory
WS2812 RGB multi-colour LED
pycom
SiPy
Bluetooth
sigfox WiFi
Model: SiPy 1.0 CE 0700
FCC ID: 2AJMTSIPY1R
External SIGFOX Antenna Connector
Reset Switch
RF Switch
U.FL Connector
Internal WiFi and Bluetooth Antenna

The PyCom company offers the **SiPy** MicroPython Wi-Fi/Bluetooth/Sigfox development board[9]. This is MicroPython compatible and designed specifically for IoT applications. It's based on the Espressif ESP32 system-on-chip (SoC) equipped with a dual-core microcontroller, Wi-Fi (range up to 1 km), Bluetooth (classic and BLE), Sigfox, and 512 KB of RAM, in a reduced footprint (55 × 20 × 3.5 mm). Programming is made easy by Pymakr, PyCom's own IDE.

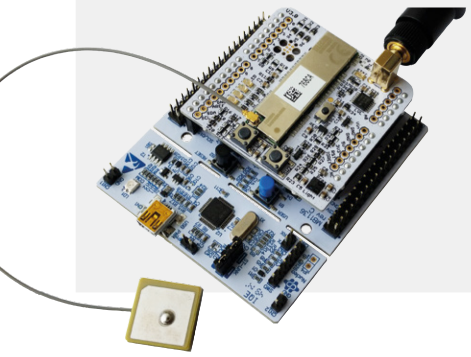
Note: An external antenna is needed for the Sigfox network. Using it without an antenna may damage the board, see RS part no. 125-9535. The Wi-Fi and Bluetooth antenna is internal.
For easy access to the SiPy pins, you can use a universal Pycom expansion board. It is fitted with a micro-USB connector for the supply and the serial communications, a LiPo battery charger, reverse polarity protection, a slot for a micro-SD card, a button, and a user LED.

Specifications

- 2 × UART
- 2 × SPI
- I²C
- I²S
- Micro-SD card
- 8-way, 12-bit ADC
- Up to 24 GPIOs
- Multicolor RGB LED WS2812
- RESET and RF switches
- TI CC1125 transceiver
- RCZ1/RCZ3 version and RCZ2/RCZ4 version

PyCom announces the launch of the FiPy WiFi/Bluetooth/LTE CAT M1 / NB1/LoRa/Sigfox [10] which, along the same lines as the SiPy, gives access to five networks.

Quicksand — STM32 development board

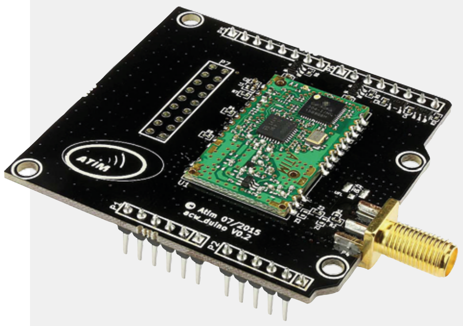


The Belgian company Quicksand is offering the QW GPS Sigfox development kit [11] built around a TD1204 transceiver and a NUCLEO-L152RE board (STM32 mbed microcontroller).

Specifications

- Sensors: GPS, accelerometer, temperature, proximity, and ambient light
- 2 user buttons, RESET button, and 4 user LEDs
- Arduino compatible via level converter
- 3.3 V and 5 V compatible
- Open-source software available
- JTAG interface to TD1204

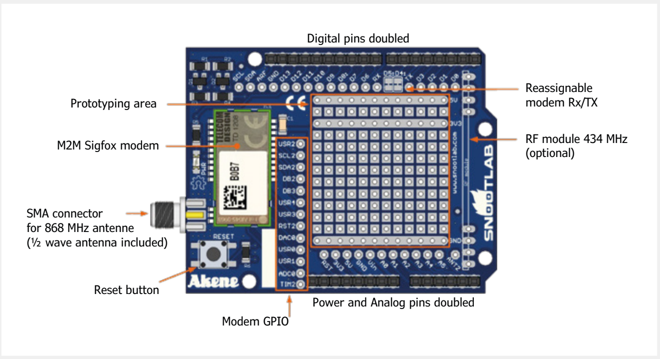
ATIM — spoilt for choice



Part no.	Adapter	Input	Output (antenna)
ACW-DUINO	Arduino	GPIO	SMA or UFL connector
ACW-MangOH	mangOH computer	GPIO, UART	UFL connector
ACW-MPCIE	mini PCI-Express	UART	UFL connector
ACW-RPI	Raspberry Pi	GPIO	SMA or UFL connector
ACW-SDK	USB (starter kit)		SMA connector
ACW-USB	USB stick	USB	SMA connector
SigBee	XBee	GPIO	SMA or UFL connector
ARM-N8-SIGFOX	Sigfox version of the ARM-NANO (Advanced Radio Modem®) radio module		

French company ATIM offers several radio and multimode (Sigfox & Lora) expansion boards [12]. The table below gives an overview of them. Most of these boards include a year's network subscription.

SNOOTLAB – Arduino and Raspberry Pi


Digital pins doubled
Prototyping area
M2M Sigfox modem
SMA connector for 868 MHz antenne (½ wave antenna included)
Reset button
Modem GPIO
Power and Analog pins doubled
Reassignable modem Rx/TX
RF module 434 MHz (optional)

The SnootLab company offers several boards using the TD1208 bridge, along with the corresponding libraries [13]:

Akeru 3.3: board compatible with the Arduino/Genuino environment. This board is also available in a kit that includes several sensors.
Akene: shield for Arduino/Genuino.
Breakout TD1208: access board for adding Sigfox connectivity to your projects, without building a special circuit for accessing the bridge signals.
Foquinha-Pi: shield for Raspberry-Pi (all versions).
All these boards are offered with a year's network subscription.

NetTrotter — 100% Arduino compatible board

This Italian company offers the BiB, Basic IoT Board [14]: this is a customizable board (addition of temperature and humidity sensors, NFC tag reading, Bluetooth Low Energy communication, etc.). As it is 100% Arduino compatible, it can take numerous Arduino shields and be programmed using the Arduino IDE. It includes a GPS and an accelerometer/magnetometer/gyroscope.

Web Links

[1] Sigfox (coverage, partners): www.sigfox.com, partners.sigfox.com

[2] Sigfox platforms for registering objects: partners.sigfox.com/products/platform

[3] Sigfox developers' website: makers.sigfox.com

[4] HIDNSEEK ST1-A tracker: www.hidnseek.fr/hidnseek_st1a

[5] GitHub for HidnSeek: github.com/hidnseek/hidnseek

[6] Tutorial for HidnSeek: github.com/Bucknalla/sigfox-hidnseek-tutorial

[7] MM002-LS-EU: www.nemeus.fr/en/nemeus-mm002-2

[8] USB MK002-LS-EU: www.nemeus.fr/en/mk002-usb-key

[9] SiPy: www.pycom.io/product/sipy

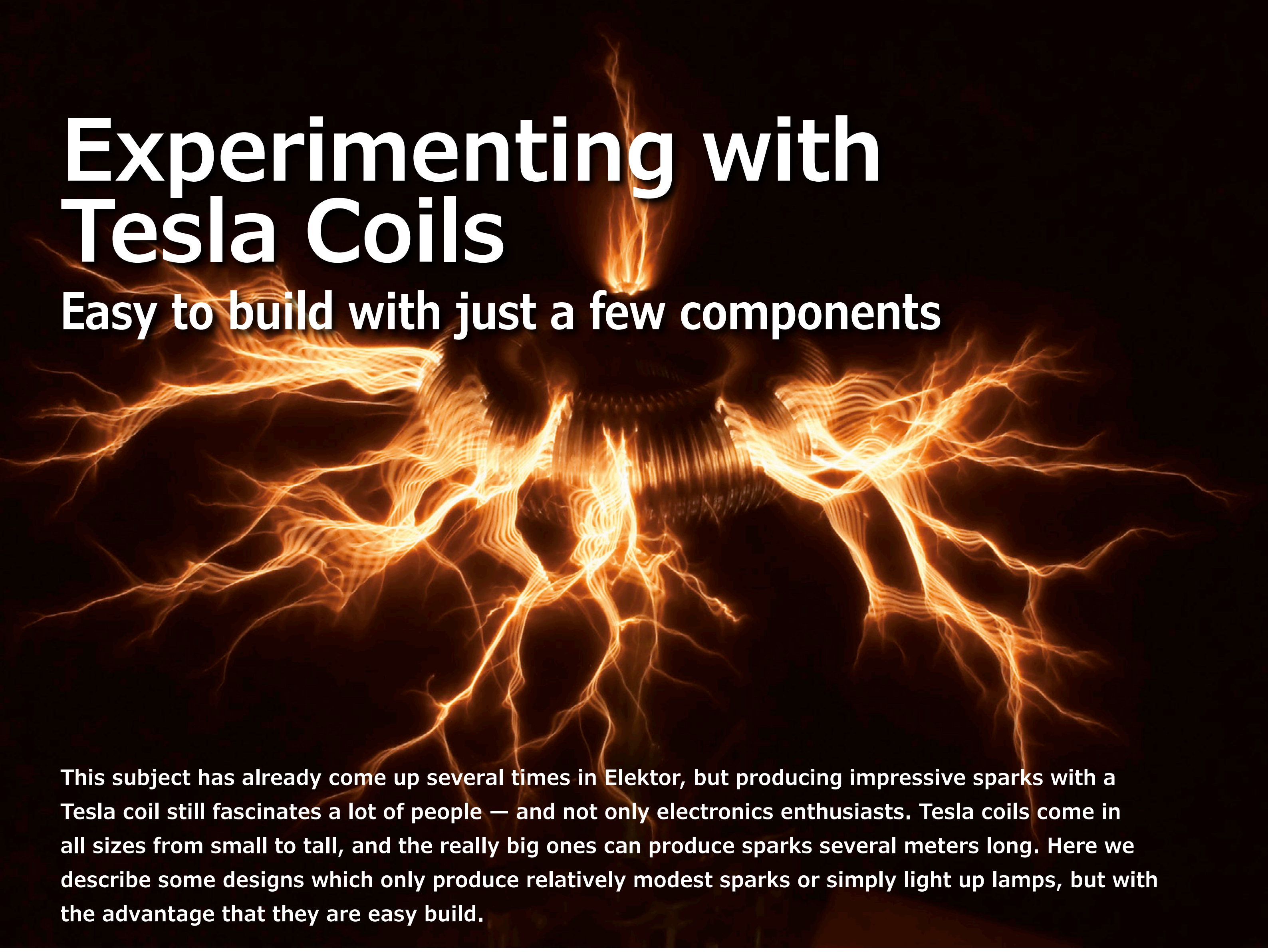
[10] iPy: www.pycom.io/product/fipy-preorder-shipping-april-2017

[11] QW Sigfox GPS: lpwan.be/wp/product/qw-gps-sigfox-development-kit

[12] ACW-DUINO, ACW-MangOH, ACW-MPCIE, ACW-RPI, ACW-SDK, ACW-USB, SigBee, ARM-N8-SIGFOX: www.atim.com/en/products/catalog

[13] Akeru 3.3, Akene, Breakout TD1208, Foquinha-Pi: snootlab.com/lang-en/72-03-iot-et-sans-fil

[14] BiB, Basic IoT Board: www.nettrotter.io/index.php/it/ecosystem-it/basic-iot-board



Experimenting with Tesla Coils

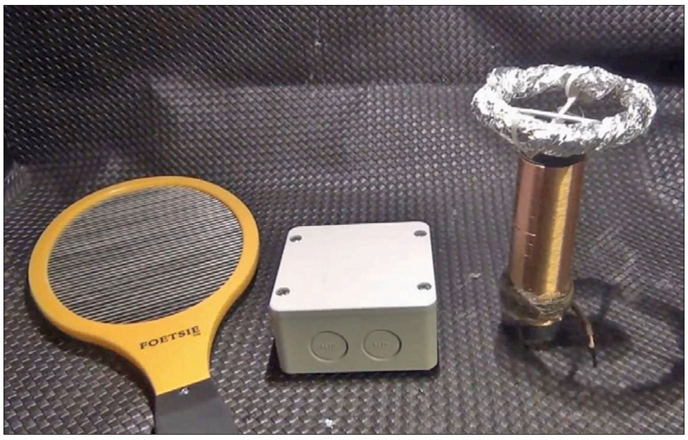
Easy to build with just a few components

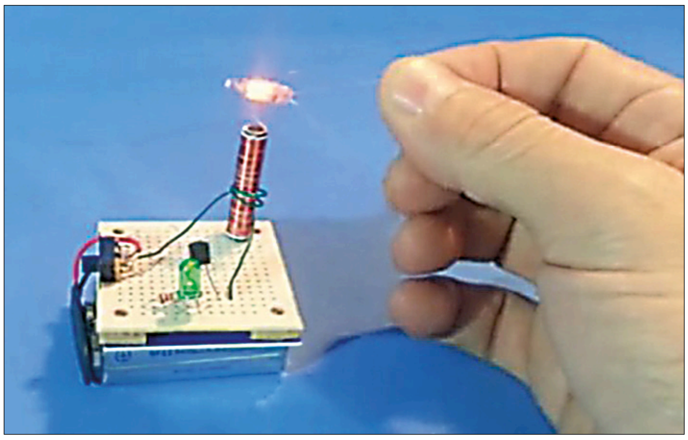
This subject has already come up several times in Elektor, but producing impressive sparks with a Tesla coil still fascinates a lot of people — and not only electronics enthusiasts. Tesla coils come in all sizes from small to tall, and the really big ones can produce sparks several meters long. Here we describe some designs which only produce relatively modest sparks or simply light up lamps, but with the advantage that they are easy build.

By **Harry Baggen** (Elektor Labs)

You have surely seen pictures or videos of devices which send lightning sparks through the surrounding air. These devices are called Tesla coils, and they are essentially air-core transformers which are able to generate very high voltages. They operate on the principle of resonance in the LC circuit of the secondary winding, which is excited by the primary winding. In the past a spark-gap circuit was often used on the primary side to generate oscillations in the LC circuit, which resulted in a lot of noise (both electromagnetic and acoustical). A semiconductor or tube circuit which drives the primary with a high frequency signal, usually between 50 kHz and 1 MHz, is mainly used with modern Tesla coils.

All Tesla coils have a characteristic form. The secondary winding consists of a long coil with a capacitive electrode mounted on the top, often with a donut shape. The breakdown voltage of air ranges from 1 to 3 kV per millimeter and is strongly dependent on the shape of the electrode (or electrodes). Breakdown occurs at a much lower voltage with a pointed rod electrode than with a spherical electrode. Assuming a conservative value of 3 kV/mm, you need a voltage of 300 kV for a 10 mm spark. If you want to learn more about how Tesla coils work, the





With a Slayer exciter you can light up fluorescent and neon lamps

Wikipedia entry on this subject [1] is a good place to start. On the hvtesla website [2] you can find an explanation with a more practical orientation, as well as some nice photos of Tesla coils in action. We can also recommend the "Tesla Coil Design, Construction and Operation Guide" by Kevin Wilson [3] as a good source of general information.

Keep it simple

Building a Tesla coil that can produce relatively large sparks is not easy. Along with a lot of mechanical work, you need a drive circuit that is suitable for high voltage and high power. However, if you just want to experiment with Tesla coils you can take a very simple approach. That's also nice for relatively small-scale experiments.

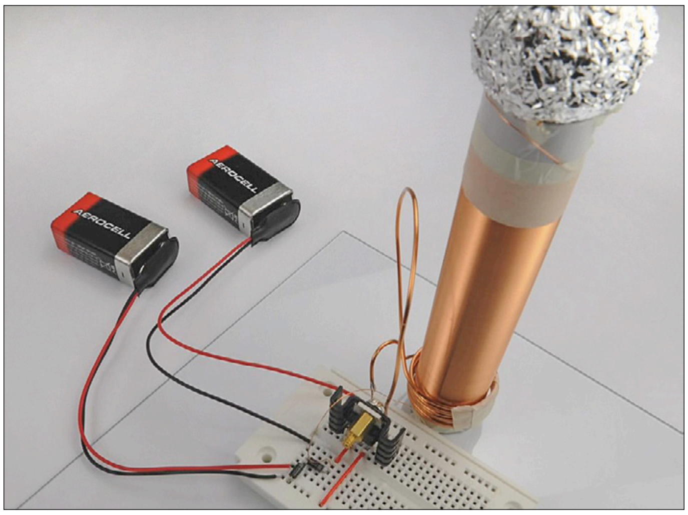
An easy way to generate a high voltage is to cannibalize a low-cost device which produces a high voltage. For example, you can start with an oscillator board from the flash circuit of a disposable camera, or a high voltage generator from an electric mosquito killer. The two video clips "How to make a small Tesla Coil" [4] show you how to build a mosquito killer HV generator into a small Tesla coil. Despite its simple construction, you can produce some nice sparks with it.

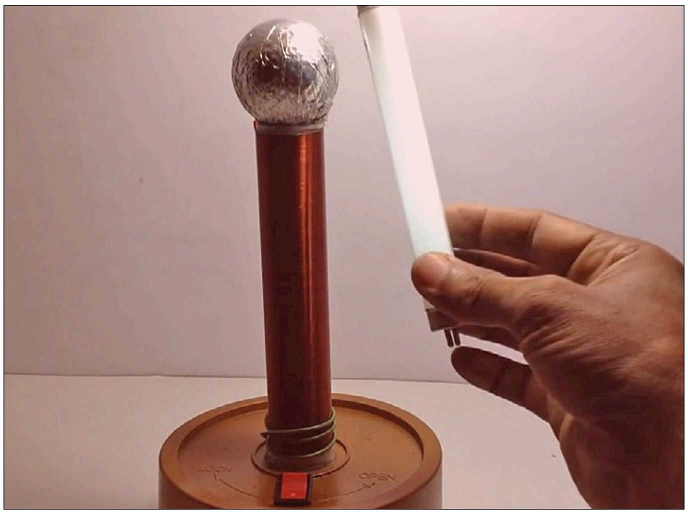
You can also achieve respectable results with a few discrete components. Most of these designs use an oscillator circuit built around a single transistor. The Learn to Discover website [5] provides a good description with a schematic. The resulting device can produce small sparks, but you shouldn't expect too much from it.

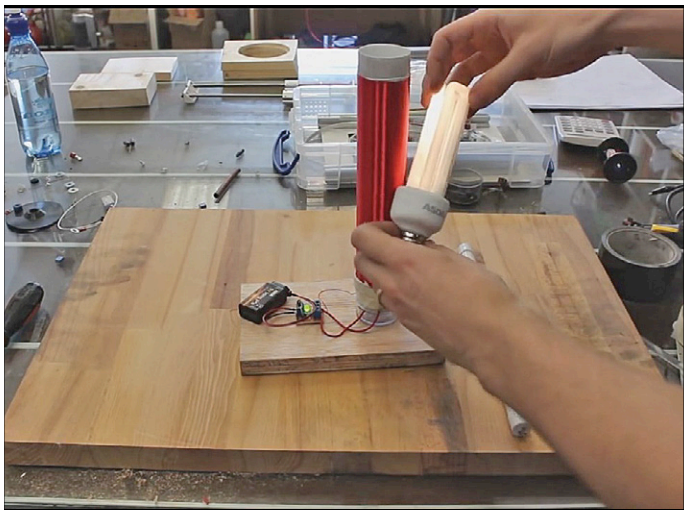
This sort of simple configuration is often called a 'Slayer exciter'. This means a circuit which generates an electromagnetic field around the secondary coil which is strong enough to light up nearby fluorescent and neon lamps, which means it is not primarily intended to generate sparks. It's also a lot of fun to play with.

If you would like to know more about this, you can have a look at the 'Miniature Tesla Coil' project on Hackaday [6]. The active component in the oscillator circuit — a 2N2222 small-signal transistor — is nothing special and the secondary coil is pitifully small, but you can still light up fluorescent tubes and lamps quite nicely with this device. There are lots of video clips on YouTube about these Slayer exciters, of which the videos at [7] are good examples. ◀

(160352-I)







Web Links

[1] https://en.wikipedia.org/wiki/Tesla_coil

[2] www.hvtesla.com/

[3] www.teslacoildesign.com/

[4] www.youtube.com/watch?v=QDZnCOLZ394
https://www.youtube.com/watch?v=X2PrPHgOy04

[5] https://learntodiscover-science.blogspot.nl/2016/10/simple-tesla-coil.html

[6] http://hackaday.com/2014/12/22/micro-tesla-coil-makes-a-perfect-stocking-stuffer/

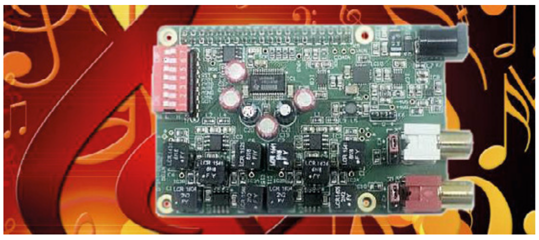
[7] www.youtube.com /watch?v=MwG-EzVEE1I
www.youtube.com/watch?v=3zjnX41K1pE
www.youtube.com/watch?v=LHCXqhhxGqA
www.youtube.com/watch?v=gSaGzhdI-QM
www.youtube.com/watch?v=gSaGzhdI-QM

Elektor Labs Pipeline

Cool projects keep being posted on the Elektor Labs website. Some are more elaborate than others; all however are interesting and made from the most treasurable commodity on earth: ~~love~~ silicon.

High-End Audio with Raspberry Pi

When you are looking for a small, stand-alone networked audio player with a touch screen you will typically find big-gun consumer amplifiers. Those that have network support either have no touch screen or are very expensive. So, why not build one yourself? Based on a cheap Raspberry Pi and a high-end digital-to-analog converter this audio DAC handles 24-bit data, sample rates up to 200 kHz, and has a built-in 8x oversampling filter.



https://goo.gl/MdGvN9

Never forget a birthday again with the Reminder Clock

Isn't it embarrassing when you miss the birthday of a beloved one? Are you one of those people who remember birthdays and anniversaries by jotting them down on Post-its or by scribbling on the new calendar every year? Come on, there must be a better way! There is, and it's called the Reminder Clock. Become the Thoughtful Parent or Friend and buy gifts or call to convey birthday and anniversary wishes "timely".

https://goo.gl/CRhzYq

Get a grip on your power consumption

Although the projects published in *Elektor Magazine* can be built and used without modifications — at least, that's what we aim for — in many cases they double as sources of inspiration. The Power Analyzer we present here is such a project. Inspired by the AC/DC Power Meter published in the September 2015 edition of *Elektor Magazine* it evolved into a 3-channel high-performance, high-precision instrument with a graphical user interface. The software was written in mikroBasic.

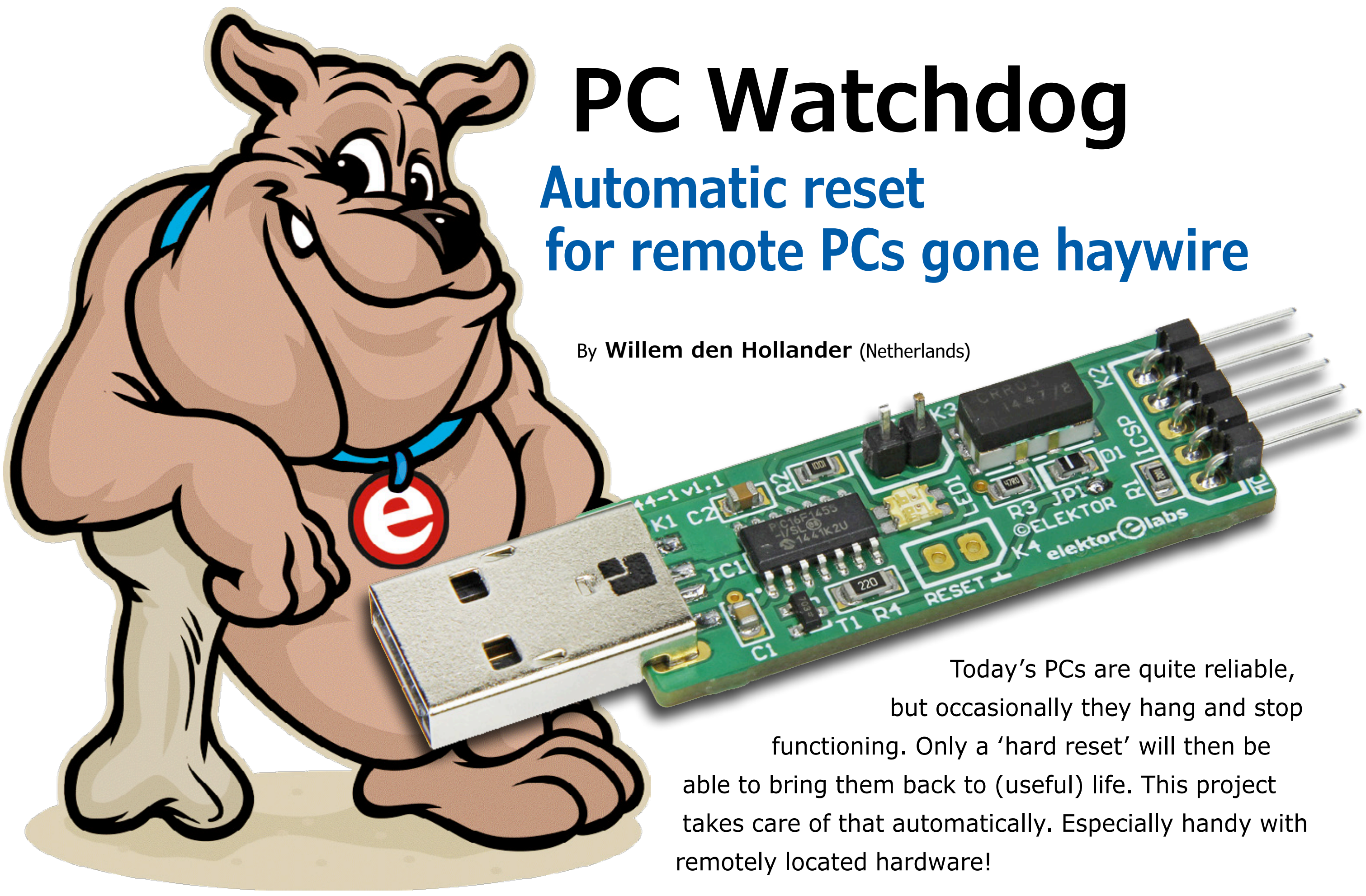
https://goo.gl/CL3pHp

Build a conductivity meter

Measuring the conductivity of chemical solution is a useful concept in lab or field operations. It can be used to monitor drinking water quality, find titrations end points, or track reaction kinetics. The Arduino conductivity shield described here is able to track the concentration of sodium chloride (kitchen salt) in water from 1.5 mg/l to about 25 g/l with a typical linearity error of 15%. This corresponds to the salinities of almost distilled water to sea water.. ⏮

(160346)

https://goo.gl/o0AipW



PC Watchdog

Automatic reset for remote PCs gone haywire

By **Willem den Hollander** (Netherlands)

Today's PCs are quite reliable, but occasionally they hang and stop functioning. Only a 'hard reset' will then be able to bring them back to (useful) life. This project takes care of that automatically. Especially handy with remotely located hardware!

While many PCs, including servers, operate unattended and/or at a remote location, it would be very useful to have a function available for sensing a malfunction of the processor and recover the system from such a situation. The realization of such a watchdog function for a PC is quite straightforward. You install a small program that generates a signal at regular intervals and have this sent to a small external circuit. Such a circuit will be able to detect when the CPU has halted by not receiving a signal from the software anymore. We then have the circuit operate the hardware reset switch to automatically reset the system.

The circuit

The schematic of the circuit is shown in **Figure 1**. Choosing a microcontroller with a built-in USB interface saves a lot of hardware. We chose the PIC16F1455. It will be recognized by the PC as an HID device. The supply voltage for the circuit is taken directly from the USB bus.
The microcontroller will receive a signal from the software running on the PC via its USB interface at regular intervals when the CPU is still running. The green section in bicolor LED1 will blink while the PC is operating normally. If the PC is bogged down and the software stops sending signals, the PIC generates a reset and the red LED (section) will light up. Right after that, while the PC is booting, the LED will light up green continuously. When the system is up and running again and the PIC receives the software signals again, LED1 will blink green.
In general the reset pin is pulled to ground to reset the PC. In most PCs USB ground is the same as ground of the reset switch, in which case the system can be reset using just one wire. This concept holds true for most PCs. Our circuit uses T1 to pull the reset pin to ground (through R4).
To rule out any doubts we added the option to use a small reed relay. The reset signal — or rather, reset short — is available on K3. When the relay is used, it doesn't matter how the two wires to the reset pens or switch are connected. The polarity of the reset switch is irrelevant as its contacts are electrically isolated. Connecting solder jumper JP1 will enable the relay. If enabled, it's best not to use K4 (output of T1).
Of course it's also possible to use the relay for other purposes. But be careful, the relay contact can only handle 170 V and 0.5 A (both DC and peak AC). It's not meant for AC line applications!

Firmware for the micro

Most microprocessors have a so-called *Watchdog* function implemented. A counter continuously counts down using either a clock generated by a separate oscillator or the main processor clock. When the counter reaches zero, a reset is generated. By regularly resetting the counter before it reaches zero using a particular instruction in the software, the watchdog is prevented from resetting the processor. The firmware for the PIC microcontroller used here can be downloaded from [1].

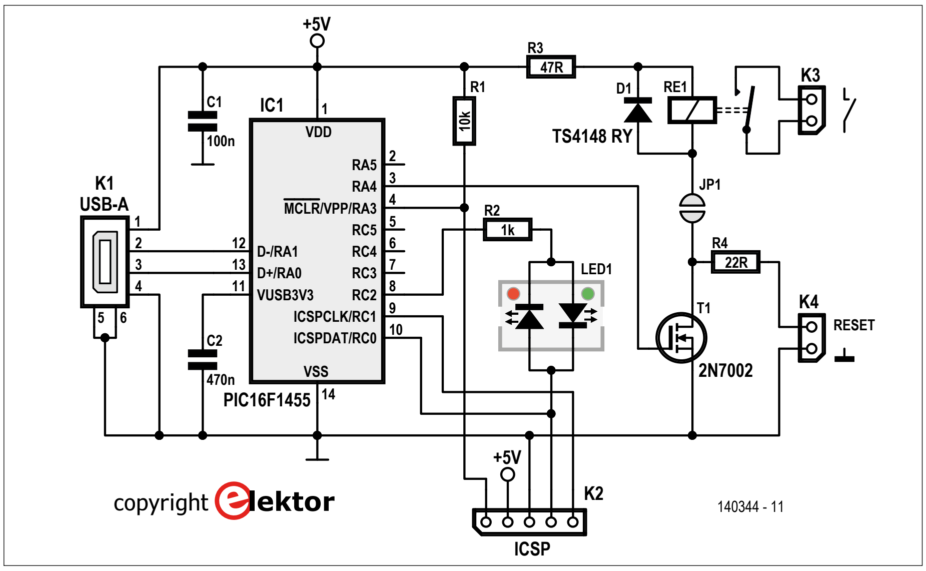


Figure 1. The schematic of the circuit. Choosing a PIC16F1455 microcontroller with a built-in USB interface saves a lot of hardware.

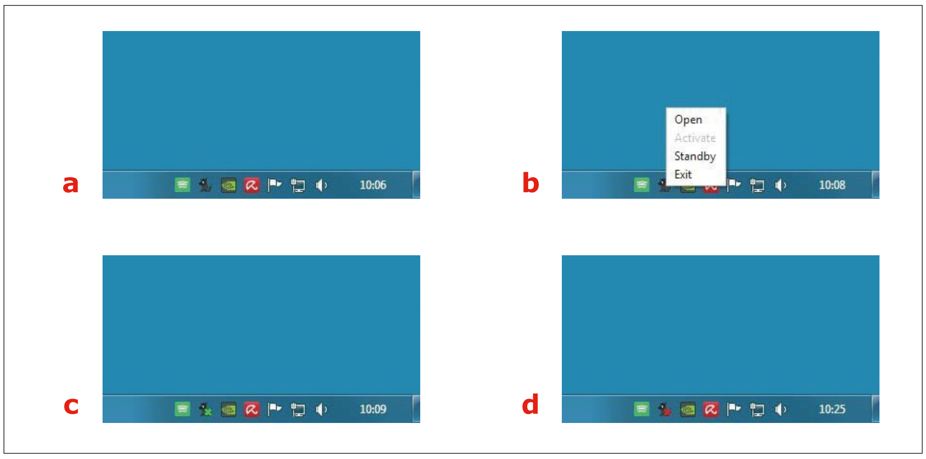
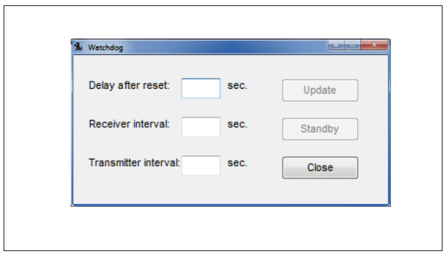


Figure 2. The little black dog icon in the Windows Taskbar indicates the status of the Watchdog software.

Software for the computer

The PC program that's part of the project needs to be started right after Windows has booted. An easy way to do this is by placing the program (or a shortcut to it) in the startup section of the list of program files (this happens automatically when the program is installed normally). When the program is running and the external circuit is connected, a doggie icon is shown on the taskbar like in **Figure 2a**, otherwise the icon is shown like in **Figure 2d**. Right-clicking on the doggie icon shows different options for the software program (see **Figure 2b**). Selecting Standby puts the external hardware in standby (**Figure 2c**) so that it will not produce a reset in the absence of an input signal. Clicking on Activate will bring it back to normal operation. The Open function, which can also be activated by double-clicking the doggie icon, will pop up the settings window (**Figure 3**). Here the various delay times can be adjusted. They will be stored in the PIC processor of the external hardware.


Watchdog

Delay after reset: sec. Update

Receiver interval: sec. Standby

Transmitter interval: sec. Close

Figure 3. Delay times can be set individually.

The *Delay after reset* option defines how the number of seconds the input signal is allowed to be absent before a reset is generated. The available range is 10 to 255 seconds. If a zero is input here, the circuit will not produce a reset signal until the first input signal puts the circuit into normal operation.

The PC will send a signal via the USB connection to the watchdog circuit at a rate defined in the *Transmitter interval* option. This number should be smaller than the Receiver interval. The Update button activates the new settings and stores them in the microcontroller's memory. The Standby button has the same function as described above, while the Close button causes the window to close. The program can also be closed by clicking the cross in the top right-hand corner.

If no input signal is received within the set *Receiver interval* after the last input signal, the circuit will activate T1 and generate a reset.

Building it

Originally the circuit was designed for connexion through a mini-USB cable. Since the circuit has to be connected to the PC's reset circuitry, a PCB is designed that can be fitted into a free USB port on the back of the computer. Doing it this way allows a wire to the reset switch (or the corresponding header on the mother-board) to remain short, and it saves another cable dangling from the back of your PC (or wherever). The PCB is designed to fit inside a small USB enclosure, which looks like a oversize memory stick (see Component List).

The reset output (K3 or K4) has to be connected in parallel with the reset switch of the PC. Probably the easiest way is to tap the wires going from the reset switch to the motherboard.

The relay is not the easiest part to solder. It can be done with hot air or, if the entire PCB is soldered in one go, in a reflow oven. The PCB shown in **Figure 4** is available from the Elektor Store as item # 140344 [2]. Excepting the relay, the parts are fairly easy to solder.

If the relay is not used then D1, R3 and K3 can be omitted. If a pinheader is to be used for K3 or K4, cut it so it won't be more than 5.5 mm above the top side of the PCB. Otherwise the PCB won't fit inside the small enclosure. The solder pads for connectors K3 and K4 are large enough to solder wires directly to. Where exactly the wire(s) leave the enclosure is up to you.

ICSP header K2 can be used to program the microcontroller, for which a PICkit 2 can be used (an update is probably called for if the microcontroller is not recognized). Horizontal header K2 is positioned in such a way that the total length of the PCB and header fit exactly in the small enclosure.

Testing it

In our first test run the watchdog persisted in resetting the PC a few seconds after the Windows logo appeared (endless loop). We tested the circuit on an older/sluggish PC with Windows XP installed (!). Files can get corrupted if the PC is reset the "hard" way (scan disk will run at reboot), so here at Elektor it wasn't something we wanted to do on a PC the IT staffers are concerned about. Changing the default settings to 0, 25 and 10 remedied it. For slower systems the difference between the settings for Receiver interval and Transmitter interval should be even larger. But even on a modern PC some programs can prevent the watchdog program from issuing a signal on time.

COMPONENT LIST

Resistors

Default: SMD 0805; 5%
R1 = 10kΩ
R2 = 1kΩ
R3 = 47Ω
R4 = 22Ω, 0.5W, SMD 1206

Capacitors

C1 = 100nF, 50V, X7R, SMD 0805
C2 = 470nF, 50V, X7R, SMD 0805

Semiconductors

LED1 = SML-LX1210IGC-TR, dual red & green LED, 160°, SMD 2.7x2 mm
D1 = TS4148 RY, SMD 0805
T1 = 2N7002, SMD SOT-23
IC1 = PIC16F1455, SMD SOIC-14

Miscellaneous

K1 = USB Plug, 2.0 Type A, right angled, SMD (2410 07, Lumberg)
K2 = 5-pin pinheader, SIL, horizontal (4-103323-1, TE-Connectivity)
K3,K4 = 2-pin pinheader, SIL, vertical (only one needed, or none, see text)
RE1 = CRR03-1A (Standex Meder), dry reed relay, SPST-NO, 3 V/70 Ω, 170V/0A5
Enclosure Strapubox USB 1, transparent, 56x20x12 mm (Conrad 531276)
PCB EPS # 140344-1 v1.1

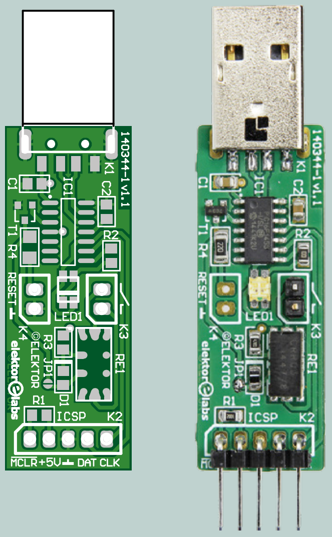


Figure 4. The small PCB is laid out to look like a USB stick.

How to test the circuit? Well "simply" have your computer crash! We first tried creating a batch file to open an infinite number of command windows in an endless loop, but that didn't work. So we used a utility we found on the internet: StartBlueScreen [3]. It initiates a Blue Screen of Death (BSOD). Works like a charm. Five command line parameters have to be specified when executing the program StartBlueScreen.exe to initiate a BSOD. Create a batch file with one line: 'StartBlueScreen.exe 0x12 0 0 0 0'. It worked. We got a blue screen and a few second later (depending where in the interval the BSOD gets initiated) the computer was reset and after booting a message appears stating the computer was reset by the watchdog.

The software for the PC was developed using Delphi XE4. Both the PC program and the microprocessor source code files are included in the free download at [1]. To install the watchdog on your PC just run 'Watchdog setup.exe'. ◀

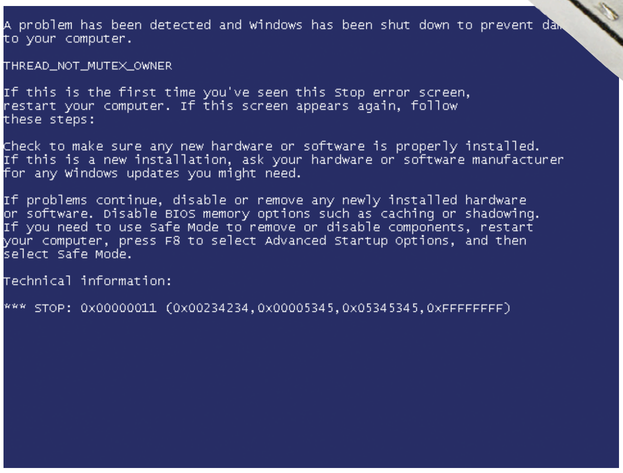
(140344)

Web Links

[1] www.elektormagazine.com/140344

[2] www.elektor.com

[3] http://nirsoft.net/utils/start_blue_screen.html


```
A problem has been detected and windows has been shut down to prevent da
to your computer.

THREAD_NOT_MUTEX_OWNER

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x00000011 (0x00234234,0x00005345,0x05345345,0xFFFFFFFF)
```

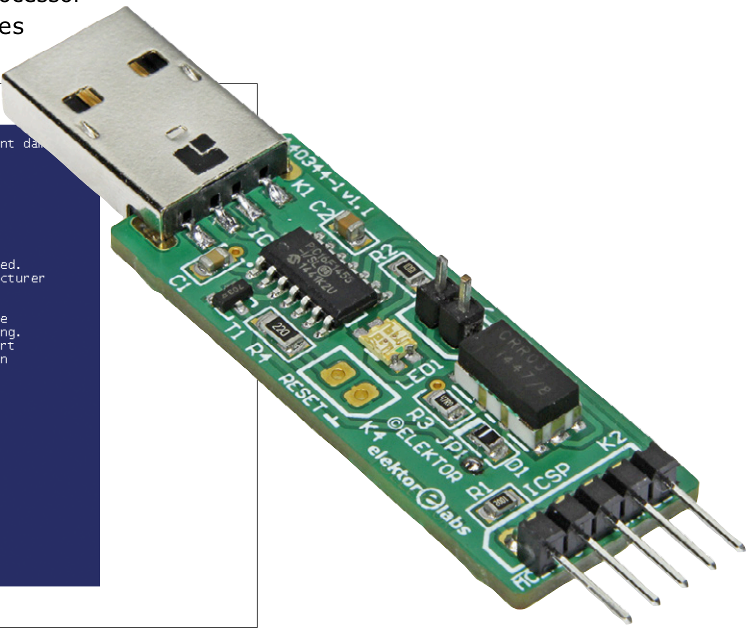


Figure 5. Probably the only time you want to see a blue screen is when you want to test this circuit.

Windows Controls Arduino (2)

Send data to your smartphone over USB or Bluetooth

By **Tam Hanna** (Germany)

Microsoft is positioning Windows 10 as a universal platform: using Visual Studio you can write a program that will run equally well on a Windows smartphone and on a PC. Beyond that, the 'Windows Remote Arduino' platform also offers simple access to real-world hardware. You can use API functions to set and clear pins and read back their values, and much besides. In the first article in this series, which appeared in the July & August 2016 issue, we showed how to send data from the smartphone to the hardware. Here we look at transfer in the opposite direction.

Before starting it is worth taking another look at the ideas we presented in the first article in this series [1]. There we saw that the 'Windows Remote Arduino' API encapsulates a protocol called 'Firmata', and thus the application programmer is spared the effort of learning the details of this protocol. API commands such as

```
myArduino.digitalWrite(13, Microsoft.Maker.
RemoteWiring.PinState.LOW);
```

are similar in appearance to the corresponding commands that would be used in an Arduino sketch to achieve the same effect: in this case, setting the level on an output pin. The library works equally well with a USB connection between the smartphone and the Arduino board and with a Bluetooth connection, and we demonstrated both with simple examples. We also saw that the system has a huge latency, which must be taken into account. If speedier operation is required, it is necessary to dispense with the library and send characters ‘by hand’, and this also entails designing your own communication protocol (see [1]).

From hardware to smartphone

In this second article in the series we will look, among other things, at how we can send data from the hardware to the smartphone.
As in the first installment we begin by installing Visual Studio. As most Elektor readers will know, this development environment is available for free download over the Internet. The source code for all the demonstration projects described here is also available, as always, for download from the Elektor website [2]. In the download the first project is called ‘WRACommunicator’.

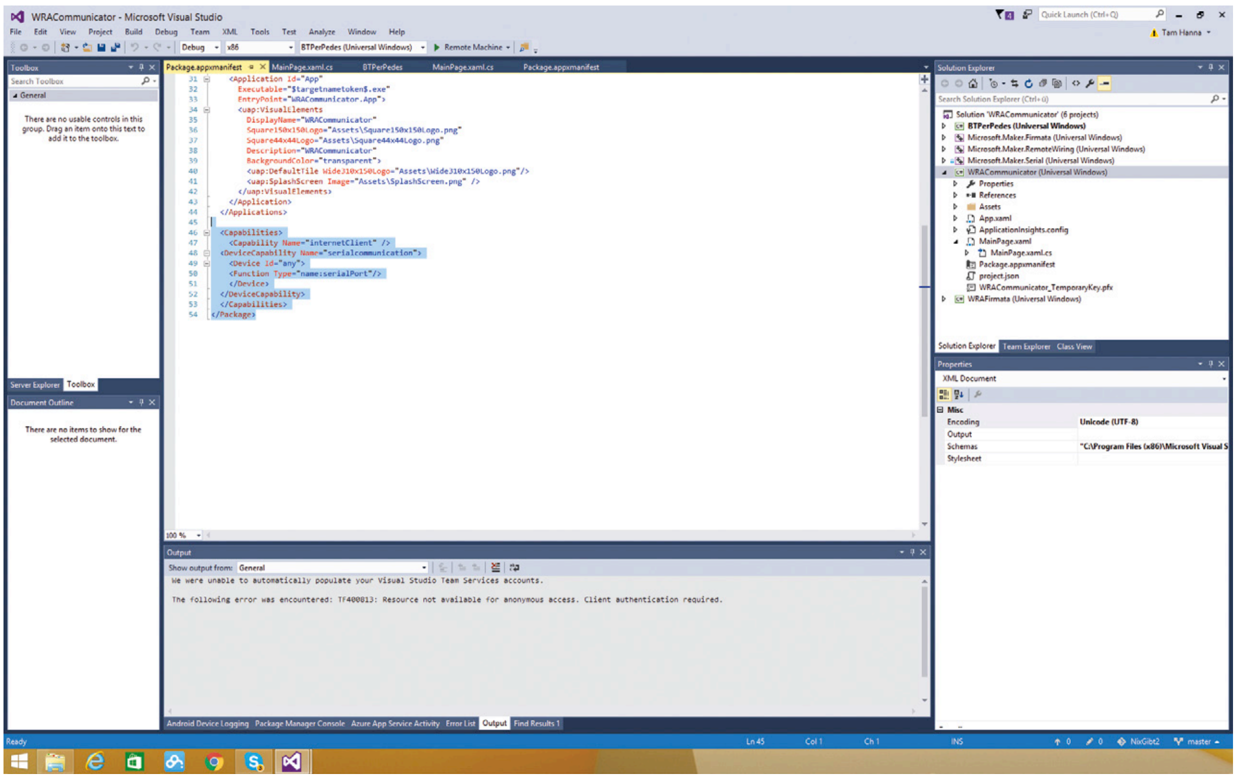
In order to use Windows Remote Arduino we need to work within a project structure, which we configure based on a standard template. Open Visual Studio 2015 and create a new project using the command *Windows → Universal → Blank App (Universal Windows)*.
The packages provided by Microsoft in NuGet (see [1]) are out of date, and in particular using `digitalRead` can lead to problems. For this reason we have to download and install the Git software version control system, available at [3].

Next start the program ‘Git GUI’, which you will find in the start menu. Select the option *Clone Existing Repository*. A dialog box will appear, and in the field labeled ‘Source Location’ enter *https://github.com/ms-iot/remote-wiring.git*.
Filling in the field labeled ‘Target Location’ is trickier. The path entered there must be a directory that does not yet exist. The author’s solution to this problem is to choose an arbitrary path with a name starting with a string along the lines of */DoesNotExistYet1*.
Check the box labeled ‘Recursively Clone Submodules’ and confirm the command. When the repository dialog box pops up close it with *Repository → Quit*.

Microsoft has divided the code into two parts; the second part can be found at the URL *https://github.com/ms-iot/serial-wiring.git*. Be careful to ensure, as before, that the directory into which you choose to clone the repository does not already exist.
After these downloads three new solutions are available. In the project that you created earlier right-click on the solution in Project Explorer, and choose *Add → Existing Project*. In the next step the following files must be added:

- Microsoft.Maker.win10/Microsoft.Maker.RemoteWiring/Microsoft.Maker.RemoteWiring.vcxproj
- Microsoft.Maker.win10/Microsoft.Maker.Firmata/Microsoft.Maker.Firmata.vcxproj
- Microsoft.Maker.Serial.win10/Microsoft.Maker.Serial.vcxproj

Open the folder *References* in the main project and then right-click on it. When you select *Add Reference → Projects → Solution*


WRACommunicator - Microsoft Visual Studio
File Edit View Project Build Debug Team XML Tools Test Analyze Window Help
Debug x86 BTPerPedes (Universal Windows) Remote Machine
Package.appxmanifest MainPage.xaml.cs BTPerPedes MainPage.xaml.cs Package.appxmanifest
Toolbox
Solution Explorer
Solution 'WRACommunicator' (6 projects)
BTPerPedes (Universal Windows)
Microsoft.Maker.Firmata (Universal Windows)
Microsoft.Maker.RemoteWiring (Universal Windows)
Microsoft.Maker.Serial (Universal Windows)
WRACommunicator (Universal Windows)
Properties
References
Assets
App.xaml
ApplicationInsights.config
MainPage.xaml
MainPage.xaml.cs
Package.appxmanifest
project.json
WRACommunicator_TemporaryKey.pfx
WRAFirmata (Universal Windows)
Properties
XML Document
Encoding Unicode (UTF-8)
Output
Schemas "C:\Program Files (x86)\Microsoft Visual S
Stylesheet
Document Outline
Ready Ln 45 Col 1 Ch 1 INS

```
    <Application Id="App"
      Executable="$targetnametoken$.exe"
      EntryPoint="WRACommunicator.App">
      <uap:VisualElements
        DisplayName="WRACommunicator"
        Square150x150Logo="Assets\Square150x150Logo.png"
        Square44x44Logo="Assets\Square44x44Logo.png"
        Description="WRACommunicator"
        BackgroundColor="transparent">
        <uap:DefaultTile Wide310x150Logo="Assets\Wide310x150Logo.png"/>
        <uap:SplashScreen Image="Assets\SplashScreen.png" />
      </uap:VisualElements>
    </Application>
  </Applications>

  <Capabilities>
    <Capability Name="internetClient" />
  <DeviceCapability Name="serialcommunication">
    <Device Id="any">
    <Function Type="name:serialPort"/>
    </Device>
  </DeviceCapability>
  </Capabilities>
</Package>
```

```
We were unable to automatically populate your Visual Studio Team Services accounts.

The following error was encountered: TF400813: Resource not available for anonymous access. Client authentication required.
```

Figure 1. Building the solution using the most recent version of Windows Remote Arduino requires us to add three existing projects.

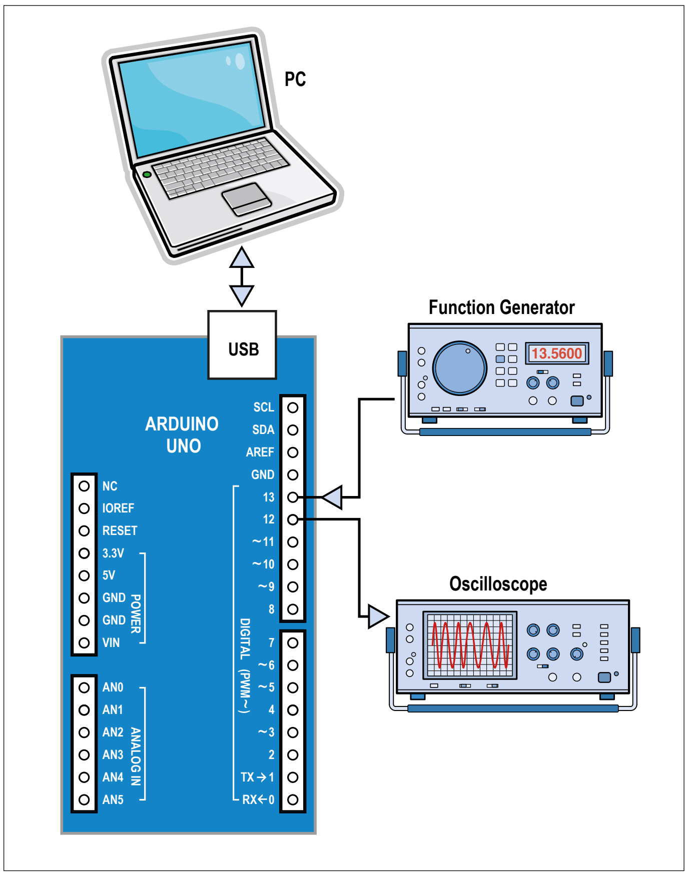


Figure 2. An ordinary signal generator can be used for test purposes.

a dialog box will appear: check all three of the projects that are shown there. That completes the integration process: the project: the overall solution should appear as shown in **Figure 1**.

Beware of the cache!

With the preliminaries out of the way we can make a start on our application. We want to send data from the Arduino to the PC and measure how speedy the transmission is. The usual way to measure this speed is to take samples of a waveform of a known frequency: from the results we can make inferences about the speed of the system under test.

Unfortunately Windows Remote Arduino uses caching in a way that makes this kind of test infeasible. Whenever Firmata detects a change in a value being read it informs the Windows device, which in turn maintains a cache of the value; and it is this locally-cached value that is read by a call to `digitalRead`. We therefore take a slightly different approach. We will 'reflect' the incoming waveform back to the Arduino, and the round-trip time will allow us to make an estimate of the speed of the system, albeit crude. We connect the Arduino Uno to a signal generator as shown in **Figure 2** and arrange for it to produce a squarewave at a known frequency. In this example we are communicating with the PC over USB, although the code shown can of course be modified to use Bluetooth instead.

As in the previous installment in this series we need to set up a manifest file and tell the Windows device the ID values of the Arduino. With that done we can begin the programming proper. As in the previous article, the function `MyUSB_ConnectionEstablished()` launches an infinite loop. When the Arduino code is started a Windows event `myArduino.DeviceReady` is triggered, and this event is connected to its own event handler, `MyArduino_DeviceReady()`.

```
private void MyUSB_ConnectionEstablished()
{
    runner();
}

async private void runner()
{
    await Task.Run(() => innerRunner2());
}

void innerRunner2()
{
    myArduino.DeviceReady += MyArduino_DeviceReady;
}
```

Why do we need to handle this `DeviceReady` event? Because only then can we start to receive signals from the Arduino. The necessary code is included in our own event handler as follows.

```
private void MyArduino_DeviceReady()
{
    myArduino.pinMode(13, Microsoft.Maker.
RemoteWiring.PinMode.INPUT);
    myArduino.pinMode(12, Microsoft.Maker.
RemoteWiring.PinMode.OUTPUT);
    myArduino.DigitalPinUpdated +=
MyArduino_DigitalPinUpdated;
}

private void MyArduino_DigitalPinUpdated(byte pin,
PinState state)
{
    myArduino.digitalWrite(12, myArduino.
digitalRead(13));
}
```

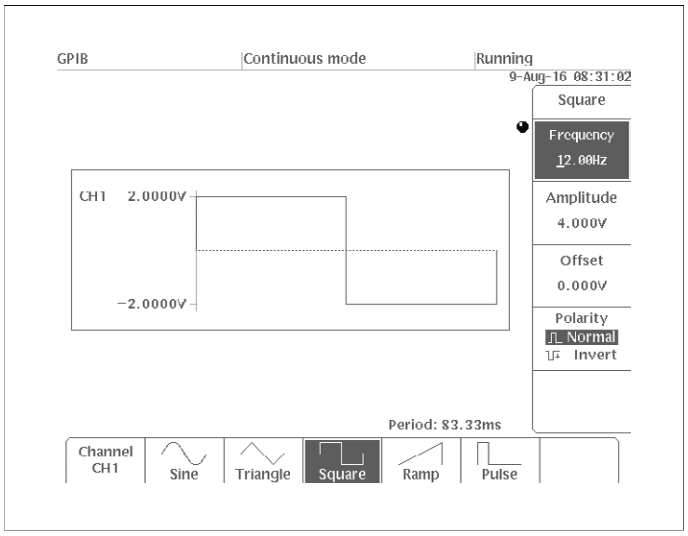

GPIB | Continuous mode | Running
9-Aug-16 08:31:02
Square
Frequency 12.00Hz
Amplitude 4.000V
Offset 0.000V
Polarity Normal Invert
CH1 2.0000V
-2.0000V
Period: 83.33ms

Figure 3. Our test waveform is not exactly challenging.

Do you see how this works? What we read in on pin 13 we immediately output on pin 12, all of course under control of the program running on the Windows device.

On the Arduino side we need the simple example sketch that can be found in the Arduino IDE under *File → Examples → Firmata → StandardFirmata*. Make sure that the communication speed is set to 115200 bps, and fire up the Arduino. Then run the program in the usual way.

The results leave us with little confidence in the real-time capabilities of the system. The waveform shown in **Figure 3** is, despite being at a frequency of only 12 Hz, seriously distorted (see **Figure 4**). At frequencies of over 20 Hz the shape of the pulses is distorted out of recognition; the effect can also be seen in the modulation domain at 12 Hz, as shown in **Figure 5**.

Expanding Firmata on the Arduino side...

We have seen that directly manipulating and reading pins entails high-latency operations. One way to make the system more efficient would be to allow addressing several pins with a single command. We will investigate this possibility with a simple example in which we address all eight output pins in one go.

As we mentioned in the first installment, the Firmata protocol is based on MIDI, and the type of each command is encoded in its first byte. In principle there is no reason why we should not extend the protocol with our own commands. In the following example we will declare a new command called DSO_QUERY. This involves adding the following line to the sketch we used above.

```
#define DSO_QUERY 0x44
```

The `sysexCallback()` method handles the commands, and so we must also extend that to suit.

```
void sysexCallback(byte command, byte argc, byte
*argv)
{
...

  switch (command) {
    case DSO_QUERY:
```

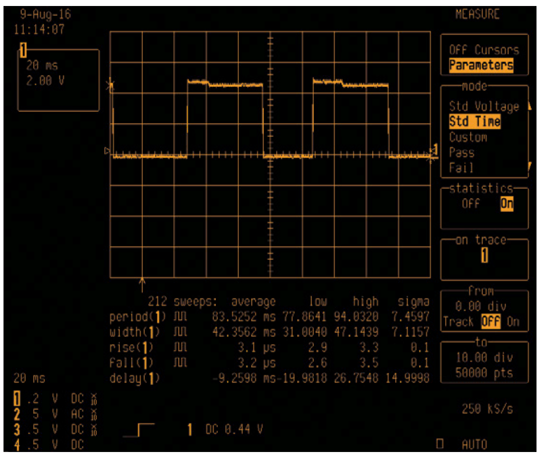
String data are processed in Firmata in a way that will be familiar to C programmers: `argc` gives the number of bytes available, and `argv` is a pointer to the beginning of the data. Our method will process the received bytes one by one, writing to the appropriate GPIO pins as follows.

```
      for(int i=0;i<argc;i++) {
        if(argv[i]==101)
          {
            digitalWrite(i+2,true);
          }
        else
          {
            digitalWrite(i+2,false);
          }
      }
```

Observant readers will be wondering at this point why the address of the pin is given by 'i+2'. The reason is simple: when serial communication is used the UART takes over digital pins 0 and 1, and so the first digital pin that is available to us is number 2.

The Firmata library also gives developers the possibility of sending messages to the controlling PC. To do this we first construct a string, which we subsequently convert to a 'CharArray' acceptable to the `sendString()` function. This array can then be sent to the host as shown in the following example.

```
sendToWin= "Got Data!";
sendToWin.toCharArray(myOutputField,64);
Firmata.sendString(myOutputField);
break;
```



```
212 sweeps:  average      low     high    sigma
period(1)   83.5252 ms 77.8641 94.0320  7.4597
width(1)    42.3562 ms 31.0040 47.1439  7.1157
rise(1)         3.1 µs     2.9     3.3      0.1
fall(1)         3.2 µs     2.6     3.5      0.1
delay(1)    -9.2598 ms-19.9818 26.7548 14.9998
```

Figure 4. There is a wide disparity between the two signal waveforms. This is a consequence of the variations in communication latency.

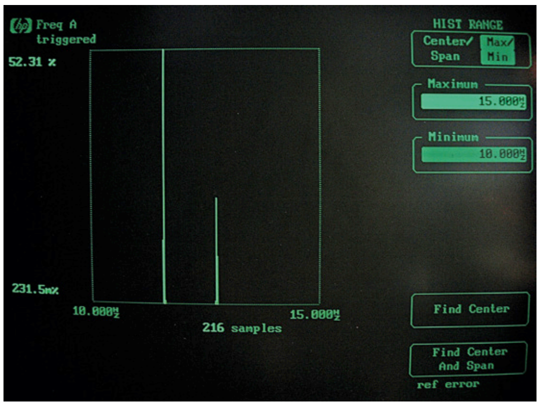


Figure 5. The jitter can be seen in the modulation domain.

In order for all the other functions of the Firmata implementation to work properly it is of course important not to touch the rest of the code!

... and on the Windows side

The addition of a new command on the Windows 10 device is rather more complicated, as we need to create our own instance of the Firmata class. The declaration of MainPage will appear as follows.

```
public sealed partial class MainPage : Page
{
    UsbSerial myUSB;
    Microsoft.Maker.RemoteWiring.RemoteDevice
myArduino;
    UwpFirmata myFirmata;
    byte DSO_QUERY = 0x44;
```

The process for initializing the communications system in this case has a completely different structure. After creating the serial communication class (here we use `USBSerial`) the next step is to create a new instance of the `UwpFirmata` class and register an event handler with it.

```
public MainPage()
{
    this.InitializeComponent();
    myUSB = new UsbSerial("VID_2341", "PID_0043");

    myFirmata = new UwpFirmata();
    myFirmata.StringMessageReceived +=
MyFirmata_StringMessageReceived;
```

A `RemoteDevice` instance can then be created using this Firmata class, and it is this which is also responsible for setting up the actual connection with the hardware.

```
    myArduino = new RemoteDevice(myFirmata);
    myFirmata.begin(myUSB);

    myUSB.ConnectionEstablished +=
MyUSB_ConnectionEstablished;
    myUSB.ConnectionFailed += MyUSB_ConnectionFailed;
;
    myUSB.begin(115200, Microsoft.Maker.Serial.
SerialConfig.SERIAL_8N1);
}
```

As in the earlier example we need to make use of the `DeviceReady` event. The first job of its event handler is to configure the required pins as outputs; it can then send a string to the Arduino as follows.

```
private void MyArduino_DeviceReady()
{
    for (int i = 2; i < 10; i++)
    {
        myArduino.pinMode(i.ToString(), PinMode.
OUTPUT);
    }
    byte[] myArray=Encoding.ASCII.
GetBytes("eaeaeaea").ToArray();
    myFirmata.sendSysex(DSO_QUERY, myArray.
AsBuffer());
    myFirmata.flush();
}
```

Meanwhile, incoming data are received in the method `StringMessageReceived()`, where they are displayed in a text box.

```
private void MyFirmata_
StringMessageReceived(UwpFirmata caller,
StringCallbackEventArgs argv)
{
    var content = argv.getString();

    Dispatcher.RunAsync(CoreDispatcherPriority.
Normal, () =>
    {
        TxtStatus.Text = content;
    });
}
```

All the code can be found in the download archive under the name 'FirmataSendReceive'.

The Bluetooth way

In view of the not exactly spectacular efficiency of Firmata, we would like to extend the direct communication over Bluetooth that we described in the previous installment by adding a reverse channel to it. For reasons of space we must limit ourselves here to sending a few 'raw' bytes and receiving them on the PC.
We will need an instance of `DataReader` in `MainPage` as follows.

```
public sealed partial class MainPage : Page
{
    ...
    DataReader myReader;
```

Establishing the connection with the device is done following the same pattern as described in [1]. The reader comprises a new instance of the `DataReader` class connected to the `InputStream` exposed by the `mySocket` class as shown in **Listing 1**.

An asynchronous method called `worker()` is responsible for the data reception proper. Its operation is relatively straightforward: when the `LoadAsync()` method is called with an `await` operator, the program will block until the requested number of bytes (in this case just one byte) is received.

Our routine reads a total of two bytes. If both of them are 255, the tautological statement `myReader=myReader` is executed. This kind of statement is useful in practice to supply a place where a breakpoint can be set. You can then easily check when data have been received by using the debugger to wait until the relevant line of code is executed (see **Figure 6**).

The sketch required on the Arduino side is not particularly complicated. Our test routine for the remote device sends two bytes every five seconds as shown below.

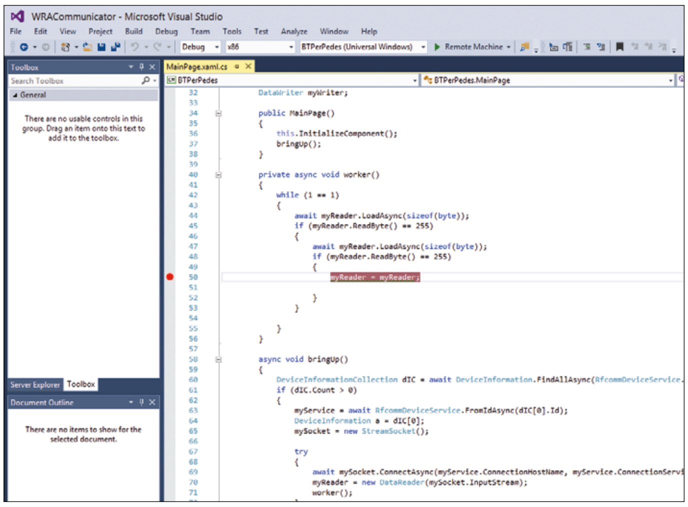
```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.write(255);
  Serial.write(255);
  delay(5000);
}
```

Before running the test program make sure that your tablet or smartphone is paired with the Bluetooth module. Then, assuming you have set the breakpoint correctly, you should find that program execution stops when the bytes are received.

Conclusion

Even taking into account its poor latency performance, Windows 10 is a highly maker-friendly operating system. As long as you are careful to delegate all the control functions to the Arduino and use Windows only for display and for accepting user input, respectable results can be achieved with little effort.

(160120)


WRACommunicator - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
x86
BTPerPedes (Universal Windows)
Remote Machine
Toolbox
Search Toolbox
General
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.
Server Explorer
Toolbox
Document Outline
There are no items to show for the selected document.
MainPage.xaml.cs
BTPerPedes
BTPerPedes.MainPage

```
        DataWriter myWriter;

        public MainPage()
        {
            this.InitializeComponent();
            bringUp();
        }

        private async void worker()
        {
            while (1 == 1)
            {
                await myReader.LoadAsync(sizeof(byte));
                if (myReader.ReadByte() == 255)
                {
                    await myReader.LoadAsync(sizeof(byte));
                    if (myReader.ReadByte() == 255)
                    {
                        myReader = myReader;

                    }
                }

            }
        }

        async void bringUp()
        {
            DeviceInformationCollection dIC = await DeviceInformation.FindAllAsync(RfcommDeviceService.
            if (dIC.Count > 0)
            {
                myService = await RfcommDeviceService.FromIdAsync(dIC[0].Id);
                DeviceInformation a = dIC[0];
                mySocket = new StreamSocket();

                try
                {
                    await mySocket.ConnectAsync(myService.ConnectionHostName, myService.ConnectionServi
                    myReader = new DataReader(mySocket.InputStream);
                    worker();
```

Figure 6. The indicated statement is useful in practice as it can form the anchor for a breakpoint.

Web Links

[1] www.elektormagazine.com/150763
[2] www.elektormagazine.com/160120
[3] https://git-scm.com/download/win

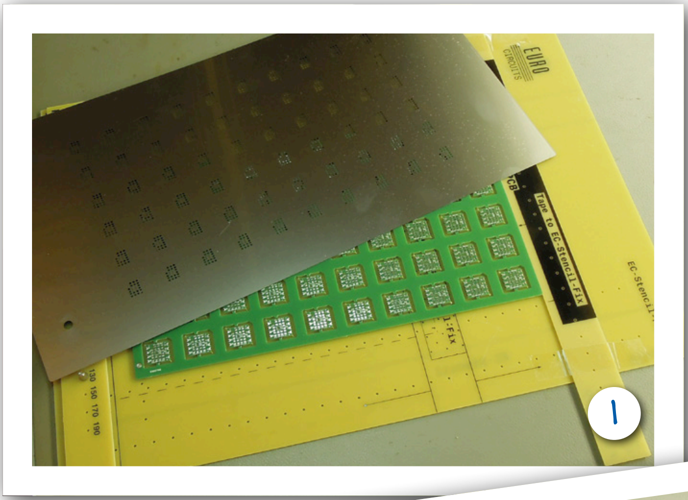
Listing 1. Processing bytes received over Bluetooth.

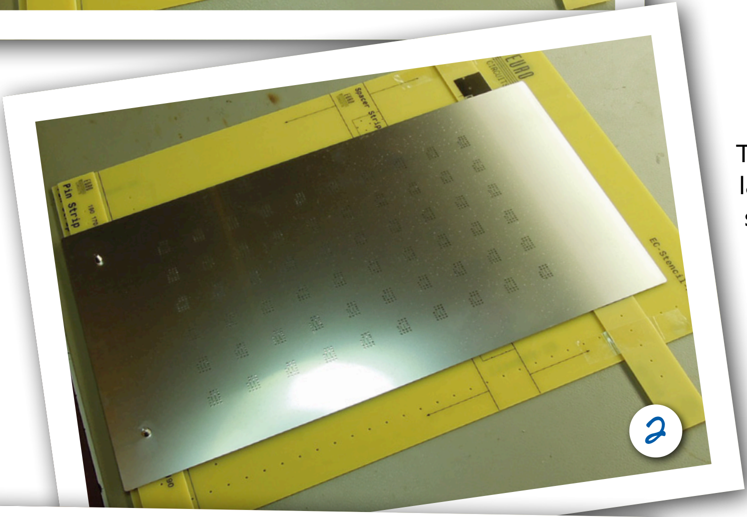
```
async void bringUp()
{
    DeviceInformationCollection dIC = await
DeviceInformation.FindAllAsync(RfcommDeviceService.
GetDeviceSelector(RfcommServiceId.SerialPort));
    if (dIC.Count > 0)
    {
        myService = await RfcommDeviceService.
FromIdAsync(dIC[0].Id);
        DeviceInformation a = dIC[0];
        mySocket = new StreamSocket();

        try
        {
            await mySocket.ConnectAsync(myService.
ConnectionHostName, myService.
ConnectionServiceName);
            myReader = new DataReader(mySocket.
InputStream);
            worker();
        }
        catch (Exception e) . . .
    }
}


private async void worker()
{
    while (1 == 1)
    {
        await myReader.LoadAsync(sizeof(byte));
        if (myReader.ReadByte() == 255)
        {
            await myReader.LoadAsync(sizeof(byte));
            if (myReader.ReadByte() == 255)
            {
                myReader = myReader;
            }
        }
    }
}
```

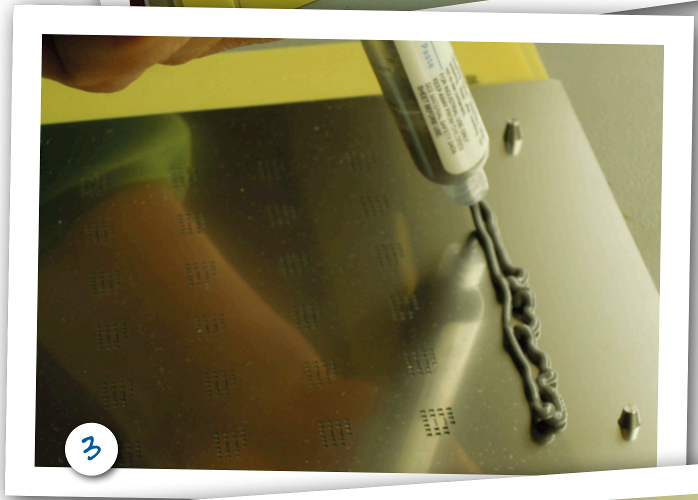
Working with the eC-stencil-fix

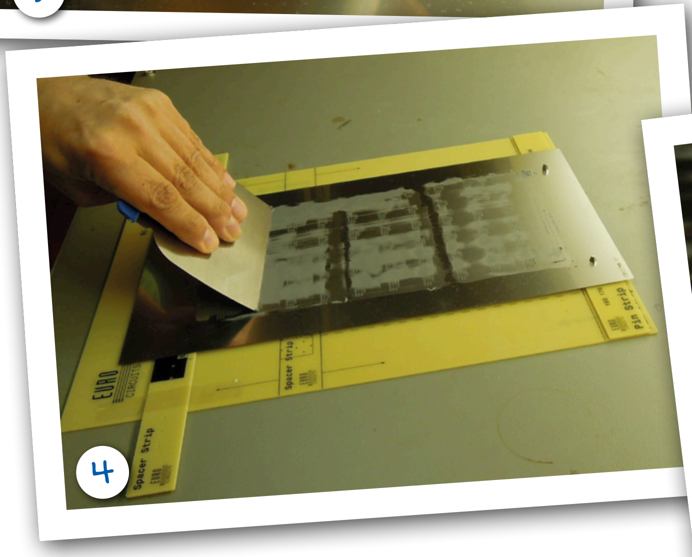
This is how we built the BME280 sensor board for you

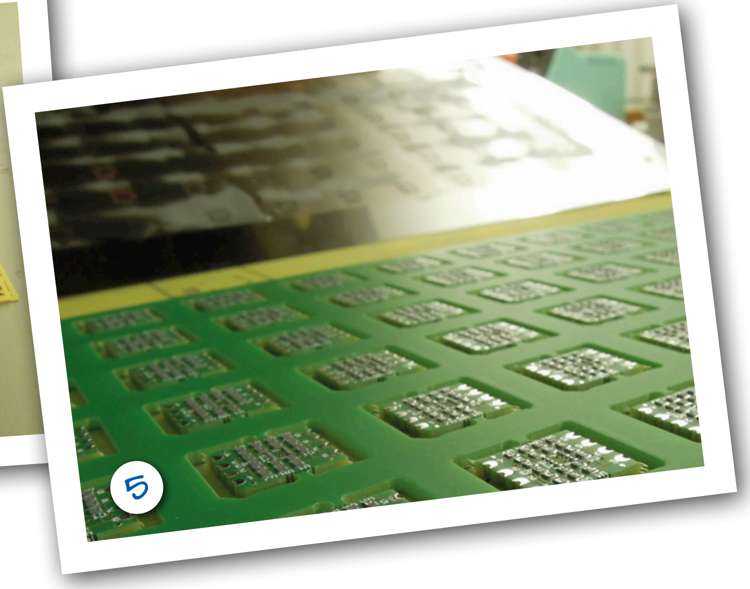
By **Thijs Beckers** (Editor-in-Chief, Elektor Netherlands)





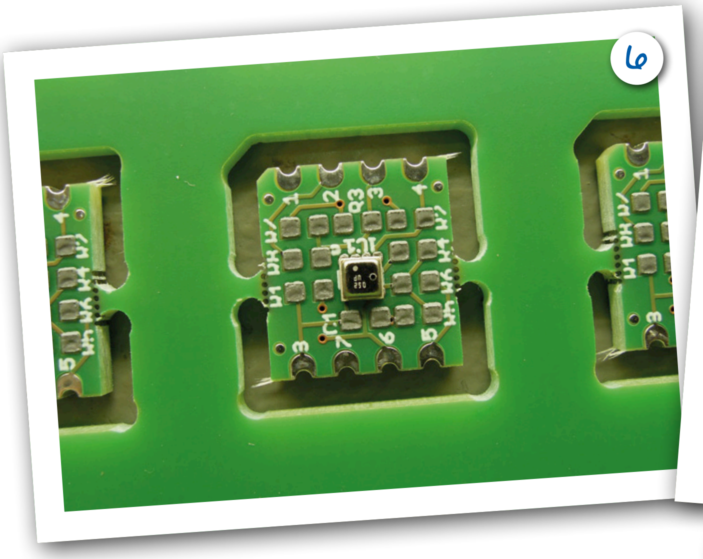


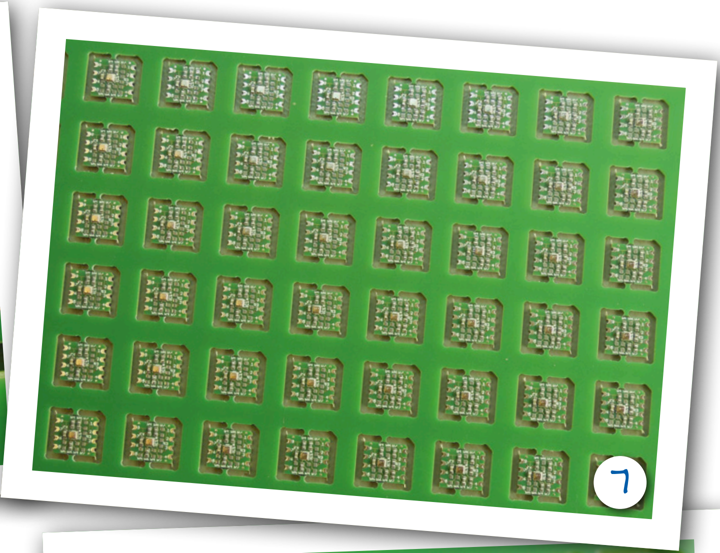


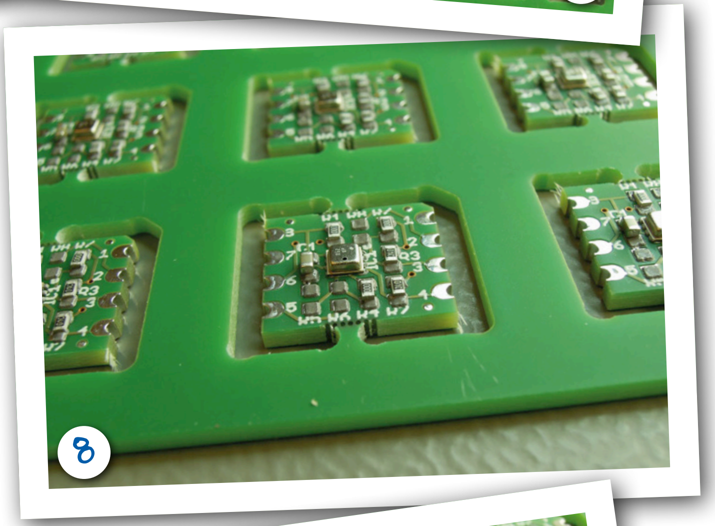


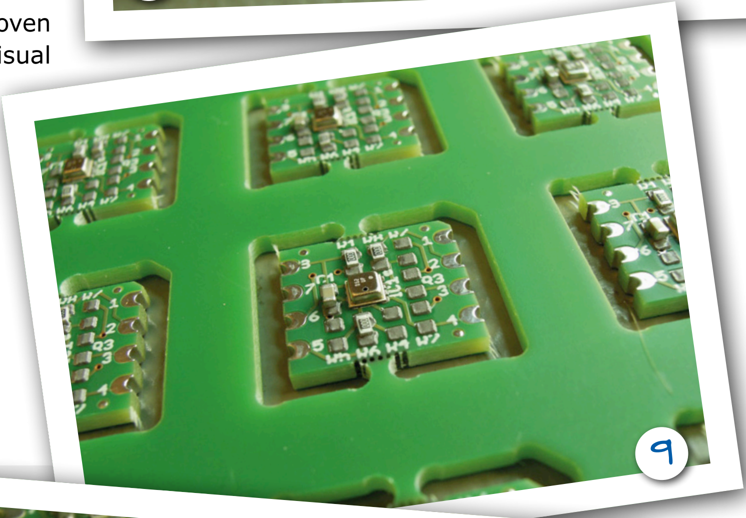
To replenish our store with products from the Elektor lab, it is sometimes faster/easier/simpler/cheaper — in short, a better option — to do the manufacturing ourselves, simply on the lab bench. This was the case for two small batches of the tiny sensor board for the BME280-shield (see Elektor March & April 2017, p. 74; [1]). And because this small sensor board would suit this perfectly, we retrieved the eC-stencil-fix [2] from the storage closet (**Photo 1**).

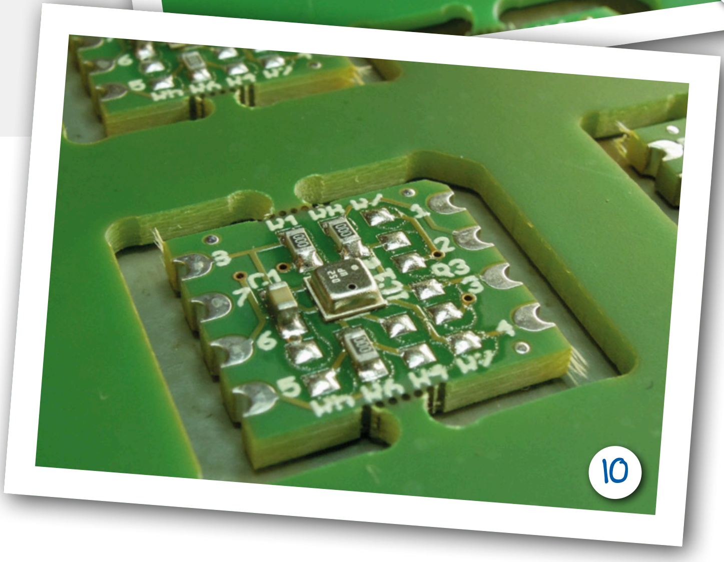
We start by correctly positioning the board and the stencil (see **Photo 2**). We locate the circuit board with two 'pegs'. During the production of our panel, Eurocircuits added special tooling holes, which are located accurately, so that it becomes very easy to position our panel. The stencil is separated from the base plate at the correct spacing using two strips of circuit board material (supplied with the eC-stencil_fix). For positioning both the stencil as well as the left-side strip we use two mushroom-shaped pegs, which go through the strip into the base plate. The right-side strip we simply tape to the base plate in the location where it slides against our PCB (see **Photo 1**). Now it is merely a case of applying sufficient solder paste to the stencil (**Photo 3**) and then spreading the paste with a suitable tool across the stencil (a spatula or plastering knife works well) so that it is deposited on top of the solder


6


7


8


9


10

pads (**Photo 4**). The thickness of the stencil is such that exactly the correct amount of solder paste is deposited on the pads (**Photo 5**).
Now we can place the components (**Photo 6**). This keeps us busy for quite a bit of time, although the board is not very large, there are quite few sensor boards in each panel with a fair number of components on each (**Photo 7**).
Incidentally, we are building two versions of the sensor board at the same time — SPI and I^{2}C (**Photos 8** and **9**). The specific version is determined by the components that are fitted. The circuit board was designed to be suitable for either version.
Now the entire assembly can be popped into the SMD oven and Bob's your uncle! (**Photo 10**). Now a good (!) visual inspection with glasses off and nose against the board to remove any faulty boards from the batch and we can make the boys (and girls) from the store happy again with new stock. The fresh sensor boards are ready to be shipped to customers. ⏮

(160297)

Web Links

[1] www.elektormagazine.com/160109

[2] www.eurocircuits.com

If you would like to know more about using the eC-stencil-fix, then we can direct you to an informative instruction video on YouTube: **https://youtu.be/HBWtqZro_fg.**

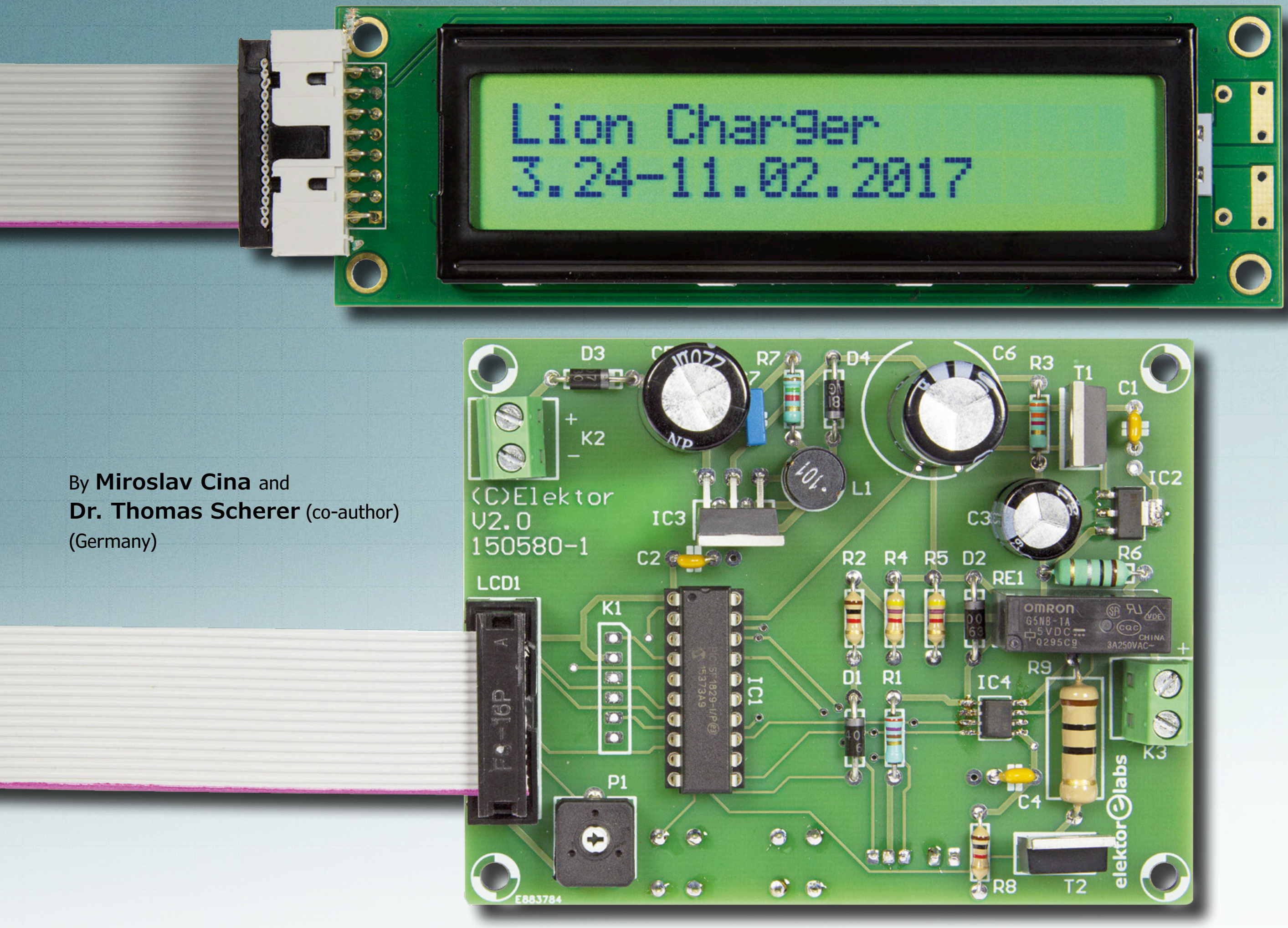
FROM THE STORE

→160109-2
PCB BME280-shield

→160109-91
BME280 sensor board (I^{2}C version)

→160109-92
BME280 sensor board (SPI version)

eLABS PROJECT



By **Miroslav Cina** and
Dr. Thomas Scherer (co-author)
(Germany)

Not so long ago the author stumbled across a bunch of old notebook batteries. They had been rejected as 'no longer fit for purpose'. Natural engineering curiosity got the better of him and armed with a screwdriver they were soon dismantled. The electronics were unimportant, and it turned out that a whole bunch of individual cells were still usable. He thought these would make ideal power supplies for his own projects. The only thing missing was a charger…

Key Features

- Charges any Li-Ion battery with a voltage rating of 3.6 V.
- Formatting mode to resuscitate cells in deep-discharge.
- Discharge mode displays the true value of battery capacity.
- All the important information is shown on an LC display.

It's not only self-build projects, but also commercial equipment which tends to favor primary batteries or NiMH rechargeables as their power source. Primary cells need to be replaced regularly while NiMH cells are quite heavy. Both types of battery are not suited to applications where high levels of power are needed and the charge needs to be retained over long storage times. For these reasons lithium-based rechargeable batteries are now becoming more common in high-end products. The technology however is more costly and requires a little more effort. Where AA / AAA cells and RC batteries in NiCd or NiMH technology can be recharged easily by low-cost charger units this is not the case for lithium batteries. The reason for this is that Li-Ion batteries are usually installed in devices (as with laptops or cordless power drill) and all the necessary charging electronics is either installed in the equipment or in the appropriate charger included with the product.
The need for universal chargers is also low, because there is no market for single lithium cells, which can be simply

Li-Ion Charger

to charge the cell correctly

inserted into devices and are sold at the checkout in Walmart or Home Depot & Co. Lithium cells are supplied in all shapes and sizes to fit inside the equipment case and they usually have solder lugs or screw terminations. They require specialized handling because, if misused, they can be dangerous.
For example, the popular LiPo cells have a particularly high power to weight ratio but are also quite vulnerable to physical damage. When damage is sustained the cell can quickly start producing smoke and flames. On top of this, standard NiMH rechargeables produce a nominal output voltage of 1.2 V whereas a single Lithium cell has a nominal output of 3.6 V. The more stable LiFePo4 types have a slightly lower value of 3.3 V but are practically never used in devices such as hand-held vacuums, power tools, smartphones and notebooks.

Li-Ion rechargeables
Regardless of whether you want to remove and use rechargeable batteries from old battery packs as shown in **Figure 1** or buy new Li-Ion cells, you will need the right charger. There is a whole lot to consider, because charging is not as straightforward as it is for other cell types such as NiMH batteries. There are many differences that need to be taken into account.

First of all, a warning: Lithium cells cannot be simply connected in series like other cell technologies such as lead, NiCd or NiMH cells without additional measures. This is because **cell balancing** is necessary. The charger described here avoids these complications because it is optimized for charging a single lithium cell with a nominal voltage of 3.6V. This nominal voltage is suitable for virtually

PROJECT INFORMATION

Lithium rechargeables
charger
Recycling rechargeables

entry level
➡ intermediate level
expert level

3 hours approx.

FTDI-USB/serial cable
PICkit-Programming interface (both optional)

€40 / £35 / $42 approx.

Figure 1. A bundle of rejected laptop batteries. Many of the cells are too good to just chuck out.

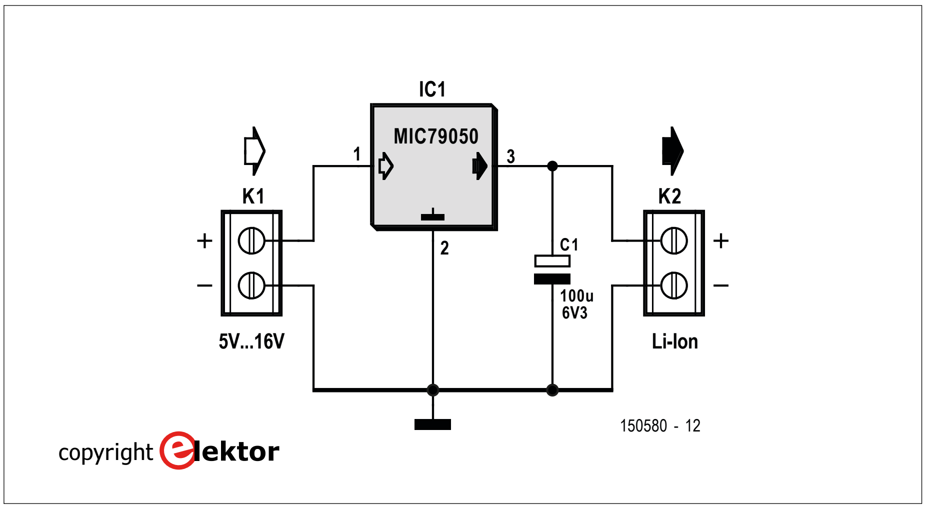


Figure 2. The minimum circuit needed to make a charger for Lithium cells uses just one three-legged IC (plus a capacitor).

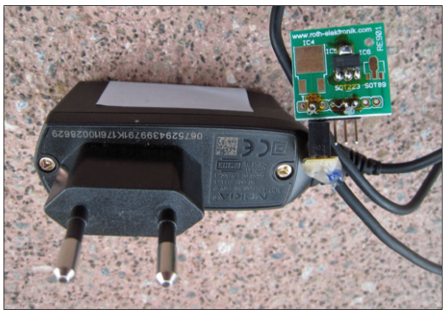


Figure 3. Together with a mains adapter Miroslav (using the circuit in Figure 2) built his first Li-Ion charger. The only drawback? You can't see what's happening.

any lithium chemistry — with the exception of LiFePo4 technology. In addition to the nominal cell voltage level of 3.6 V, there are some other 'typical' levels that must be taken into account when building a charger. Depending on the cell type, a minimum voltage of 2.75 to 3 V and a maximum voltage of 4.2 V must be observed when using the cells. When the cell voltage is allowed to go outside these levels during cell storage, charging or discharging then you must expect some damage to occur to the cell.
It's worth noting that when 'recycling' individual cells you won't necessarily be able to achieve the full specified cell capacity. A cell that's not in its first flush of youth may still be able to store enough charge to be useful in many applications. It is also possible to try to recover a cell that has suffered a deep discharge. In such cases you need to be careful, don't blast it with a full charge current. With a cell voltage less than 2.7 V, you first need to 'format' them with a minimum current of a few mA. If the cell rises above 2.7 V it's safe to charge at the normal rate. This process may take several hours. When a cell doesn't even manage to reach this threshold after several hours, then the only recycling option left is the old battery collection point at the supermarket. The same goes for cells that measure a stable 0.0 V or are slightly negative (occurs with cells in series circuits). There's no way back for these so don't waste your time.

Charge regulation

An important property of a rechargeable lithium battery is its '*C*' value. You may be tempted to think this stands for 'capacity' but rumor has it that a certain Monsieur Coulomb may have some connection here. It is the value of charge or discharge current in relation to the cell's rated capacity. The formula is:

C = capacity in Ah / 1 h.

A charge current of 1*C* for a battery rated at 2,300 mA/h will be 2.3 A. Charging at this rate with a constant current equal to 1C is perfectly possible for many cells. Some specialist cells or older cells can only cope with about 0.5 *C* or a bit less. Fast-charge types can handle up to 5 *C* and are fully charged after 20 minutes. Charging at a lower value of current than the rated maximum is of course always possible.
This factor also plays a role in discharge, but higher rates can generally be achieved. 2 *C* is the typical maximum, the Panasonic 18650 cell used in the original Tesla cars are probably loaded at this level. Some cells however are even better in this respect: LiPos often quote 5 *C* or 10 *C* and some special cells can even deliver more than 20 *C* for a short time. For charging, however, this plays a less important role.

The requirements for a lithium charger can therefore be directly derived from these parameters: It must provide a charge of only a few mA when the cell is below 2.7 V, above this threshold it is then charged at a higher constant current until the final voltage of 4.2 V is reached. From there, the voltage of 4.2 V is maintained and the charging current is automatically reduced with increasing 'fill level' of the cell. If the resulting charge current at any time drops below a minimum value of, say 0.05 *C*, the battery is considered full and the charge current is disconnected. NB: Excessively damaged batteries may be so leaky that the charge current never drops below the threshold to trigger disconnection. In this case a timer function is used to prevent continuous charging.

An integrated solution

Lithium batteries are usually already installed inside the equipment they power; it is no surprise that semiconductor manufacturers already have a wide range of integrated charger controllers in their product lines to make life easier for equipment manufacturers. One of the simplest integrated charging solu-

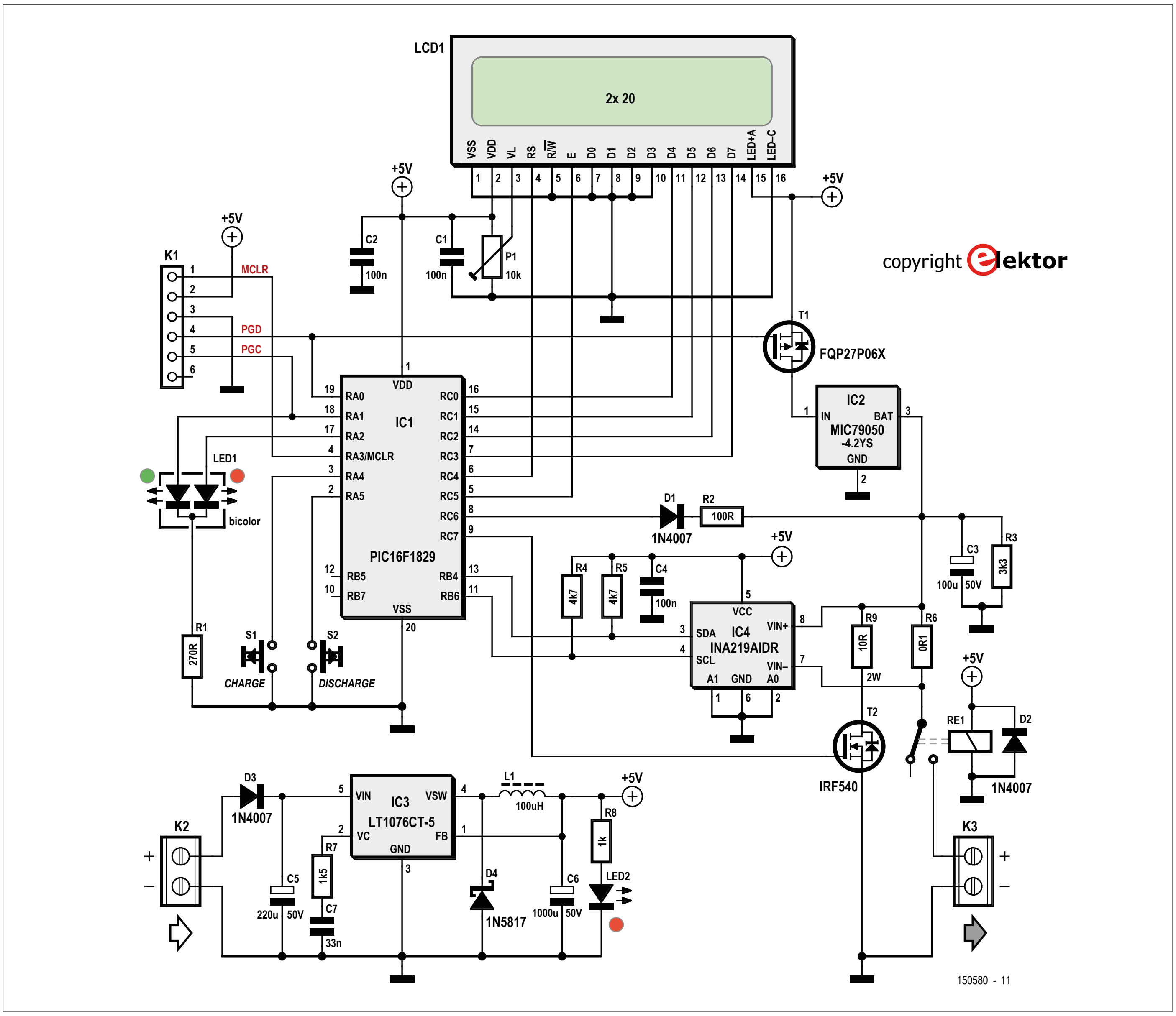


Figure 4. With the help of a microcontroller the simple circuit shown in Figure 2 is transformed into a luxury charger.

tions is the three-pin IC MIC79050S from Microchip. A charger designed around this chip can be as simple as the one shown in **Figure 2**. This charge regulator provides a stabilized voltage of 4.2 V with a (required) tolerance of 0.75 % at its output. Its charge current is a maximum of 0.5 A — not much, but this level is suitable for the vast majority of cells. A normal 2 to 3 Ah cell will be charged in 4 to 6 hours which is acceptable.
Apart from a built-in over-temperature protection, does not have many other features. It's available in the solder-friendly SOT223-3 housing and requires nothing more than an input voltage in the range 5 to 16 V at 0.5 A. It's recommended to keep the input voltage at the lower end of its range otherwise the energy dissipated in this linear regulator will generate too much heat. Miroslav built a simple prototype lithium cell charger using an old Nokia mobile phone plug-in power supply (5.7 V / 0.8 A) together with this IC on an experimental board (see **Figure 3**). The first test proved successful and gave him the green light to carry on.

This simple charger design does the job well but has just one disadvantage for electronics enthusiasts: we like to know what's going on, the battery is charging but we can't see any information and don't have a clue how long before it's full. This requires a little more electronics, which is why Miroslav developed the 'luxury' version of the charger.

The luxury version

When you talk of 'more luxury' these days it usually means that a microcontroller has been installed to add some extra bells and whistles. That's also the case here: **Figure 4** shows the complete circuit of a convenient charging device for commercially available Li-Ion cells. The PIC16F1829 microcontroller not only controls charging but measures the voltage and current levels, controls a two-color status LED, polls two push buttons and displays all the important information on an LCD.

But it doesn't end there: To prevent too much energy being wasted as heat, a switching regulator IC3 is included in the circuit. You don't need to worry about winding a coil for this; it uses a standard off-the-shelf inductor L1. D3 provides reverse polarity protection. The regulator IC3 requires an input voltage between 5 V and 60 V to produce the 5-V output. This means that a simple, unstabilized plug-in power adapter with a capacity of about 5 W is enough to power the charger circuit. Any voltage between 9 V and 36 V should do. An ideal adapter would supply 12 V at 0.5 A. D4 should be a fast Schottky diode rated at 1 A. Don't be tempted to substitute a standard silicon diode you found at the bottom of your spares box. LED2 illuminates when 5 V is available from IC3.

IC1 polls the status of the two pushbuttons S1 and S2 with I / O pins RA4 and RA5. The input network is a bit Spartan because the internal pull-ups of these controller inputs are activated. RA1 and RA2 drive the two-color LED (see table **Bi-Color-LED and Status**). LCD1 uses a classic half-byte data access and consists of two lines of 20 or 24 characters depending on the display type. P1 controls the display contrast. The backlight is connected directly to +5 V (pin 15) and ground (pin 16). Note that the specified display has a series resistor built-in for the LED backlight. If you choose to use a different display without a built-in resistor you will need to add one externally in series with the backlight supply pin. T1 is a P-channel power MOSFET which switches the charge regulator IC2 on and off.

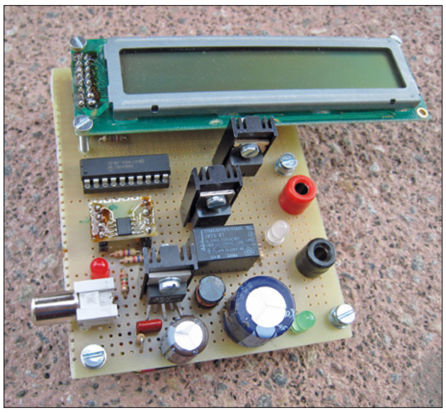
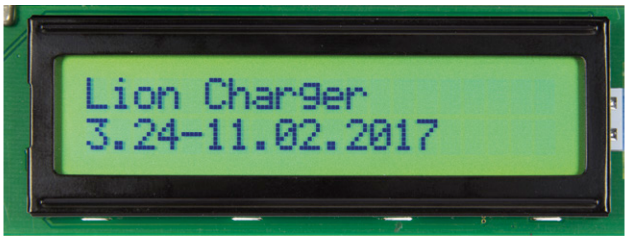
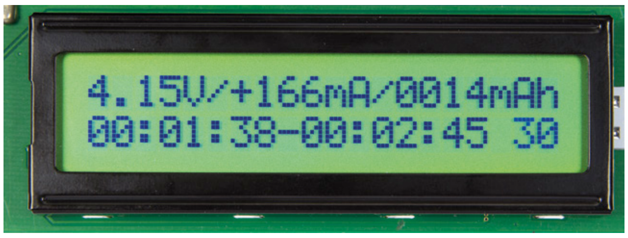
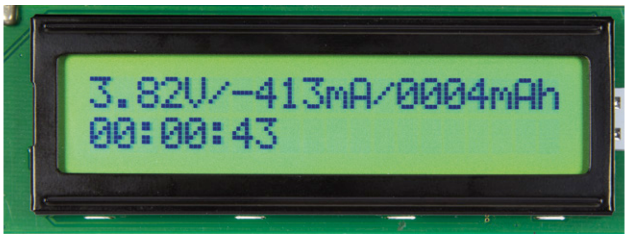


Figure 5. This is Miloslav's perfboard prototype of the luxury charger.


```
Lion Charger
3.24-11.02.2017
```


```
4.15V/+166mA/0014mAh
00:01:38-00:02:45 30
```


```
3.82V/-413mA/0004mAh
00:00:43
```

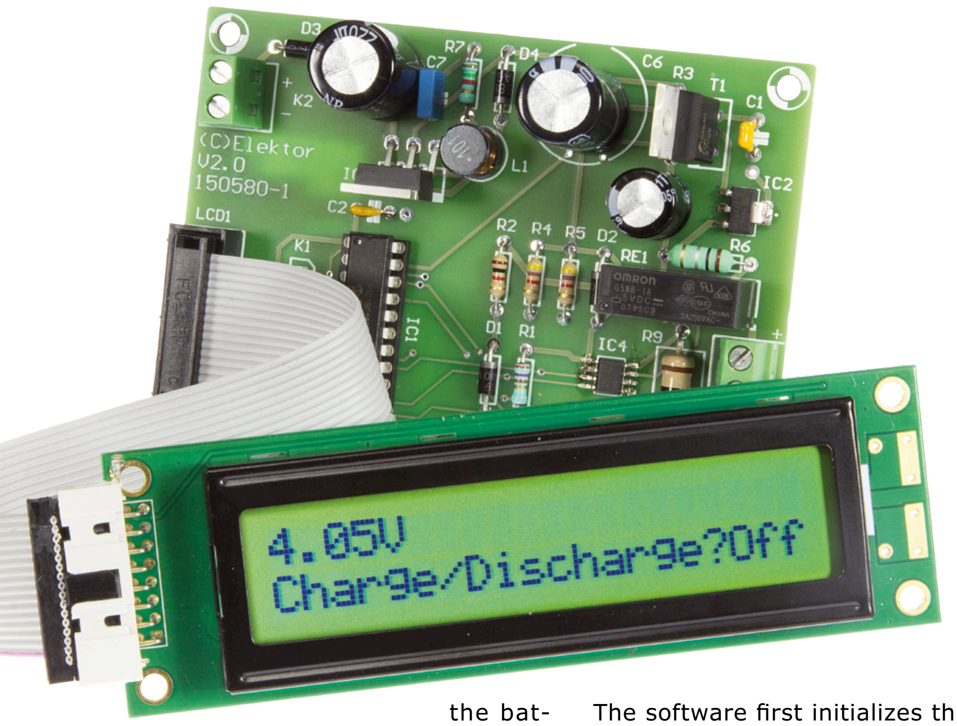
The current measurement is carried out by IC4 (INA219), which is specialized chip for measuring current using a low-impedance shunt (R6). IC4 communicates with IC1 via an I²C interface. For this application the least precise version of the INA219B is used. It's more than accurate enough and is the least expensive version of the chip.

Transistor T2 connects the load resistor R9 down to ground, providing a discharge path for the battery. In its discharge mode (activated by pressing S2) the true capacity of a (fully charged) test battery can be determined completely automatically. The output pin RC6 of IC1 provides a low level trickle charge to 'format' a battery that is too deeply discharged. The current is limited by R2 and the internal resistance of the MCU pin, D1 prevents reverse current damage. The level of this formatting current depends on the cell voltage. For example: with a battery level of 2.3 V and the pin supplying 5 V minus the forward voltage of D1 = 2 V across R2. This equates to a maximum current of 20 mA, the MCU cannot supply more than 25 mA.

It only remains to clarify, what exactly relay RE1 is actually doing, it's not controlled by the microcontroller. The relay coil is connected directly to the supply voltage. The contacts will only close when the charger has power. If the charger is unplugged with the battery still connected the relay disconnects

Balancing

Unlike the chemistry used in other types of rechargeables, lithium batteries are very sensitive to overcharging and deep discharging, which will produce irreversible damage. Manufacturing tolerances ensures that each cell produced has a slightly different value of capacity. During charging of several cells connected in series the cell with the smallest capacity in the chain will always get "over-charged" (the cell voltage rises above its permissible maximum) while the other cells still want to be charged further. In addition to the temperature rise and possible fire hazard, the damage to the affected cell will increase and reduce its capacity even further. The same is true for discharging: without monitoring each cell, the one with the smallest capacity will be the first to fall below the critical undervoltage threshold. The damage again leads to a further drop in capacity and after several charge / discharge cycles, the problem with the cell gets worse. So the golden rule is that lithium cells must never be connected in series without a so-called balancer or some dedicated battery management electronics to monitor their state. This is true for normal operation (= discharging) as well as for charging. Suitable balancers are available for different cell counts, capacities and cell chemistry. The issue of balancing is complex and beyond the scope of this article. There are already articles in Elektor, dedicated to the subject which can be found by searching www.elektormagazine.com. For many applications, however, the use of a single cell alone will suffice; that way you can safely ignore the tricky balancer issues.



the battery and prevents it discharging 'backwards' through IC3 and IC4, reliably preventing unintentional deep discharge.

Software

No microcontroller without firmware. In this application it performs the following tasks: Determine the current state, measure current and voltage, poll the pushbuttons, drive LED1, T1 and T2, record the time and output relevant information on the LC display. The firmware was written in assembler and is available free of charge both as a source code and as a hex file [1]. The following is a brief overview of relevant parts.

The software first initializes the time counter (with interrupts and TIMER2); Then the main loop is entered.
The *charge_main* routine takes care of the charging process. In the main loop, the actual values of current and voltage are read and evaluated via I²C communication with the INA219. If the cell voltage is below 2.7 V formatting will be initialized. T1 is now turned off. If the voltage rises above 2.69 V formatting is terminated and the normal charging process is started via T1 and MIC79050.

During the normal charging phase, voltage and current are measured periodically. The main loop is terminated when the charging current drops below 80 mA or when the total maximum charging time of 30 hours is exceeded. Both values can be changed in firmware. The value for the maximum charge duration is in the *mm02* subroutine at the position:

```
movlwD'030' ;maximum charging
 time (hours)
```

The value for the minimum charge current is in the *curr_an* subroutine at the position:

```
xorlwD'080' ;keep charging above
 080 mA
```

It is also possible to change the threshold for switching from high to low charging current in the same subroutine.

```
xorlwD'200' ;green LED off above
 200 mA
```

The *discharge_main* subroutine takes care of the discharge phase. It stops when the cell voltage drops below 3 V. During the discharge time is not measured, but the current delivered into the load is measured = capacity in mAh. There is no timeout. The actual measurement is calculated in the subroutine *inc_q* in mAh.

Operation

After **power up** the display briefly shows the firmware version followed by information about the rechargeable battery connected. The information shown here

Bi-Color LED and Status

LED1 indicates the operational status of the Li-Ion charger. The following indications apply:

- LED1 = **off**. The charger is not active. The battery cannot be charged or discharged.
- LED1 = **orange** (R and G both lit). Cell formatting in progress.
- LED1 = **red**. Charging phase. The charge current is > 200 mA.
- LED1 = **green**. Charging phase. The charge current is 80 to 200 mA.
- LED1 = **red blinking**. Discharge phase.

When the charger is first turned on with the battery connected nothing happens, just LED2 lights to show that power is available. By pressing S1 LED1 will begin the charge process, when the cell voltage is below 2.7 V the charger attempts to format the cell and the LED will be orange. When the cell voltage is above this threshold the LED is red and the cell is charged with a current >0.2 A. When the current falls below 0.2 A the LED turns green and when the charge current falls below 80 mA the LED goes off and the charge cycle is finished.
When S2 is pressed, discharging begins and the LED keeps flashing red until the cell voltage drops to 3 V. All processes can be stopped at any time by pressing both pushbuttons at the same time.

appears on the 2 x 20 LCD:

```
U = 3.10V
CHARGE? / DISCHARGE?
```

Now press S1 to start charging or S2 to start discharging.

When **Normal Charging** (cell voltage >2.7 V) the LCD will display this information:

```
4.16V/+203MA/1736MAH
05:02:29           25
```

In the first line the value of cell voltage and current are given as well as the total current flow in mAh. The second line gives the elapsed time and remaining time until timeout (25 hrs shown). As the current drops below the **200 mA threshold** the display changes to:

```
4.16V/+199MA/1758MAH
05:05:29-00:05:24   25
```

In the second line now we can see an additional timer indicating elapsed time at this lower current level.

When the cell is **completely charged**, LED1 will go off and the display will, for example be showing:

```
4.16V      1807MAH
05:05:29-00:10:320FF
```

In the second line at the end, instead of timeout we can see the information 'Off'.

Should the cell voltage be too low to start a normal charge cycle the charger will switch to **formatting mode** with the following display:

COMPONENT LIST

Resistors
(All 5%, 0.25W, 250V)
R1 = 270Ω
R2 = 100Ω
R3 = 3.3kΩ
R4,R5 = 4.7kΩ
R6 = 0.1Ω, 1W
R7 = 1.5kΩ
R8 = 1kΩ
R9 = 10Ω, 2W

Capacitors
C1,C2,C4 = 100nF, 50V, ceramic, 0.2" pitch
C3 = 100µF 50V, radial, 3.5mm pitch, ø 8 mm
C5 = 220µF 50V, radial, 5mm pitch, ø 10 mm
C6 = 1000µF 50V, radial, 7.5mm pitch, ø 16 mm
C7 = 33nF 50V, ceramic, 0.2" pitch

Inductor
L1 = 100µH, 190mΩ, 900mA, radial, MCSCH895-101KU

Semiconductors
D1,D2,D3 = 1N4007
D4 = 1N5817 *
LED1 = bicolor, 5mm, common cathode
LED2 = red, 3 mm
T1 = FQP27P06X, P-channel, 70mΩ
T2 = IRF540NPBF, N-channel, 44mΩ
IC1 = PIC16F1829-I/P, programmed*
IC2 = MIC79050-4.2S, TO223-3
IC3 = LT1076CT-5
IC4 = INA219AIDR

Miscellaneous
LCD1 = LCD, with background light, e.g. MC22005A6W-GPTLY from Farnell *
6-pin pinheader, straight, 0.1" pitch
2 PCB mount 2-way terminal block, 0.2" pitch
2 PCB mount pushbuttons, N/O, 6 x 6 mm
Relay, 5V, 1-pole, N/O, 5A, e.g. Omron G5NB1A5DC
20-pin IC socket, DIP, for IC1
PCB 150580-1 V2.0 *

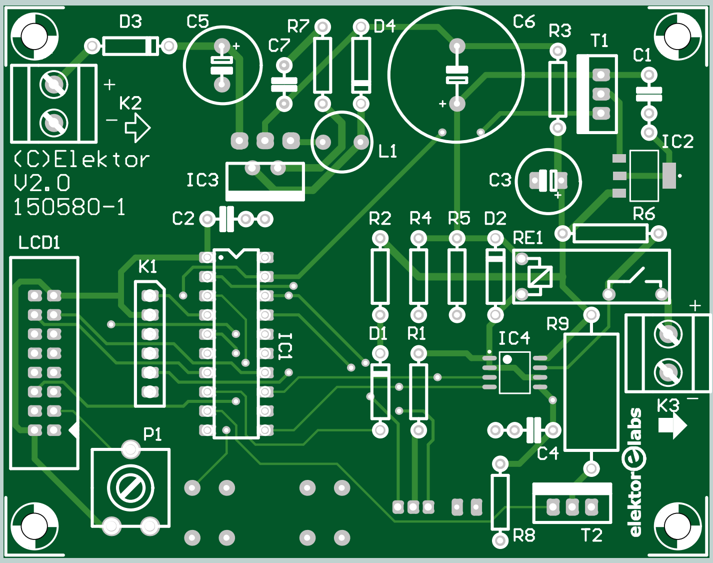
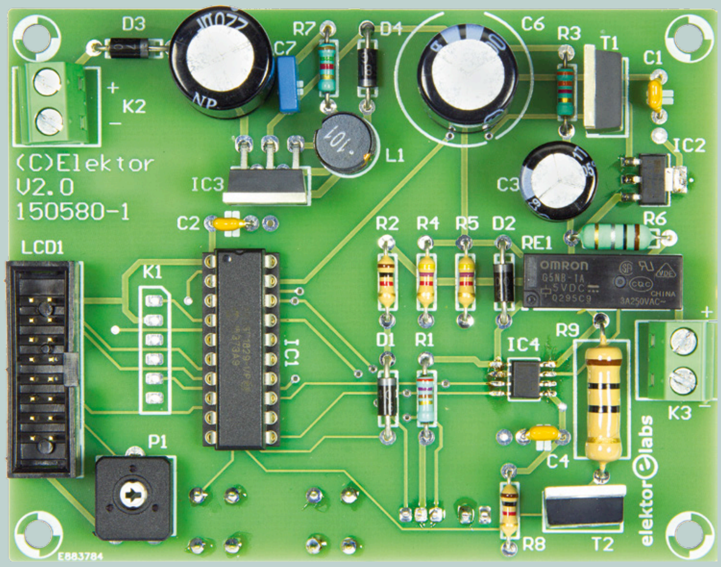


Figure 6. The PCB makes building the Lithium charger luxury version child's play.



* see text

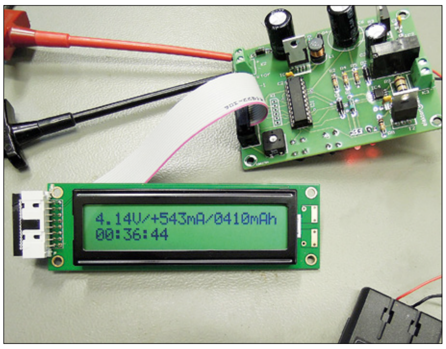


Figure 7. Prototype from the Elektor lab using the PCB shown in Figure 6.

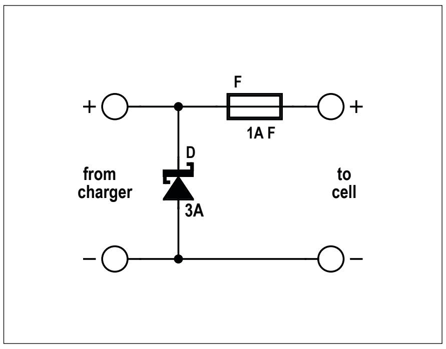


Figure 8. This simple protection circuit prevents an incorrectly connected battery from damaging the charger.

FROM THE STORE

→150580-1
PCB

→150580-41
Programmed Controller, version for 2 x 20 characters

```
1.28V/+016MA/0000MAH
FORMAT  00:01:43 30
```

In addition to the actual cell voltage and charging current the total current value will be very low. The second line indicates the mode and how long the process has been running and how long the charging process may take.

When **Discharging** by pressing S2 the energy in the battery will be dissipated in resistor R9. A fully charged battery will initially show a discharge current of around 360 mA. The display will look like this:

```
3.72V/-364MA/0759MAH
00:46:10
```

In first line after the actual values of cell voltage and current (negative sign = discharging) the value of current discharged in mAh is given. The second line shows the elapsed time.

Construction and final comments

Miroslav's prototype laid out on a square of perfboard is shown in **Figure 5**. Built using the PCB (**Figure 6**) available to order from the Elektor Store [1] it looks so much neater and the build is not difficult. All components except IC4 (which is also not too difficult to solder) are through-hole type components, making them easy to solder. IC1 is pre-programmed (with the firmware to drive a 2x20-character display).
If you want to use a display with 2x24 characters, the alternative firmware can also be downloaded. If you do use an alternative display, you should ensure that it has an HD44780-compatible controller. Once all the components are fitted the final assembly should look like the prototype from the Elektor lab (**Figure 7**).
When connecting the battery be careful. A fully charged battery connected to the charging terminals the wrong way round can damage IC4, the chip can only withstand a low negative voltage at its measuring inputs. If you want to ensure this situation cannot occur the protection circuit shown in **Figure 8** can be used between the battery and charger. When the battery is connected the wrong way round the diode conducts and blows the fuse. Use a fast-blow fuse and make sure the diode is a Schottky-type (rated at 3 A should be enough) to keep the reverse voltage applied at the input as low as possible and therefore avoid damaging the charger.
Since the construction of his Li-Ion charger, Miroslav has used lithium cells to replace many the NiMH batteries fitted in equipment around the home. They offer better performance and longer time between recharges. The charger is therefore often in use and has proved itself reliable in practice.
For questions, suggestions and ideas for improvements, Miroslav is available at
miroslav.cina@t-online.de.

(150580)

Web Link

[1] www.elektor.com/150580

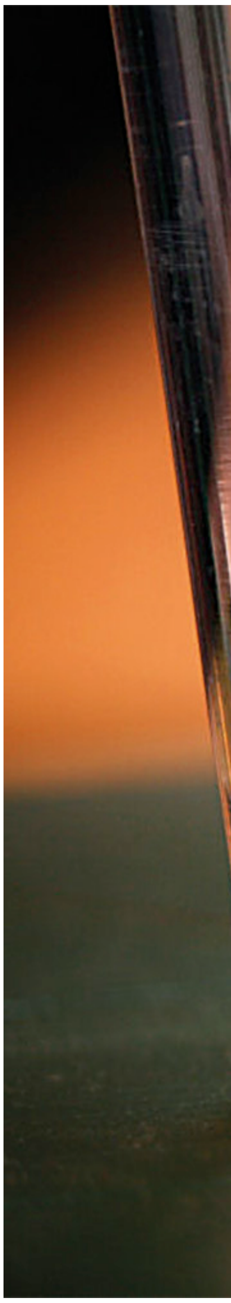
Advertisement

www.panavise.com
EXCLUSIVE
UNVEILING
of the newest PanaVise Circuit Board Holder designed for Makers
See it at...
MAKER FAIRE
BAY AREA
2017
May 19-21
NEW
PANAVISE®
Innovative Holding Solutions

ELEKTOR ETHICS

Do We Need a Robot Law?

European lawmakers call for EU-wide rules to govern the ethical and legal aspects of robots and artificial intelligence. Should these emergent technologies be regulated while they are still being developed? Or should they be allowed to mature first before being scrutinized by legislators?



By **Tessel Renzenbrink** (Netherlands)

Last February the European Parliament (EP) adopted a resolution in which it calls on the European Commission to draft a legislative framework to regulate robotics and AI. The resolution consists of some 19 pages of recommendations covering what such a framework should aim for. These comprise a wide range of areas including innovation, standardization, ethical principles, employment & education and liability rules. In more detail, this is what the Parliament states about these topics:

Innovation

Europe is a frontrunner in AI and robotics R&D. To keep it that way the EU should increase research budgets and promote open science & responsible ethical innovation. Secondly, a solid digital infrastructure lies at the basis of the development of these technologies. The EU should therefore aim for ubiquitous connectivity & universal access and uphold the principle of net neutrality. And lastly, interoperability between systems is essential for real-time data flows that enable robots and AI to become more flexible and autonomous. Open standards and transparency should therefore be promoted in order to avoid lock-in in proprietary systems that restrain interoperability.

Standardization

To stimulate healthy competition in the fields of robotics and AI, standardization and interoperability are key. The EU should therefore work on the international harmonization of technical standards in collaboration with the European and International standards bodies. Closer to home, EU-wide rules should avoid fragmentation of the internal market and prevent the development of incompatible systems in the different Member States. Makers and inventors will be happy to learn that the EP stresses the importance of lawful reverse-engineering and calls for tinker-friendly zones where experiments with robots are permitted.

Ethical guidelines

Asimov's Laws of Robotics cannot be converted into machine code, therefore they should be understood to be directed at designers, engineers, producers and operators of robots, the EP states. It proposes a code of conduct for all those active in the field that would commit them to four basic principles:

1. robots should act in the best interests of humans;
2. robots should not harm a human;
3. humans should have the autonomy to make an informed, un-coerced decision about the terms of interaction with robots;
4. the benefits of robotics should be fairly distributed in particular those of homecare and healthcare robots.

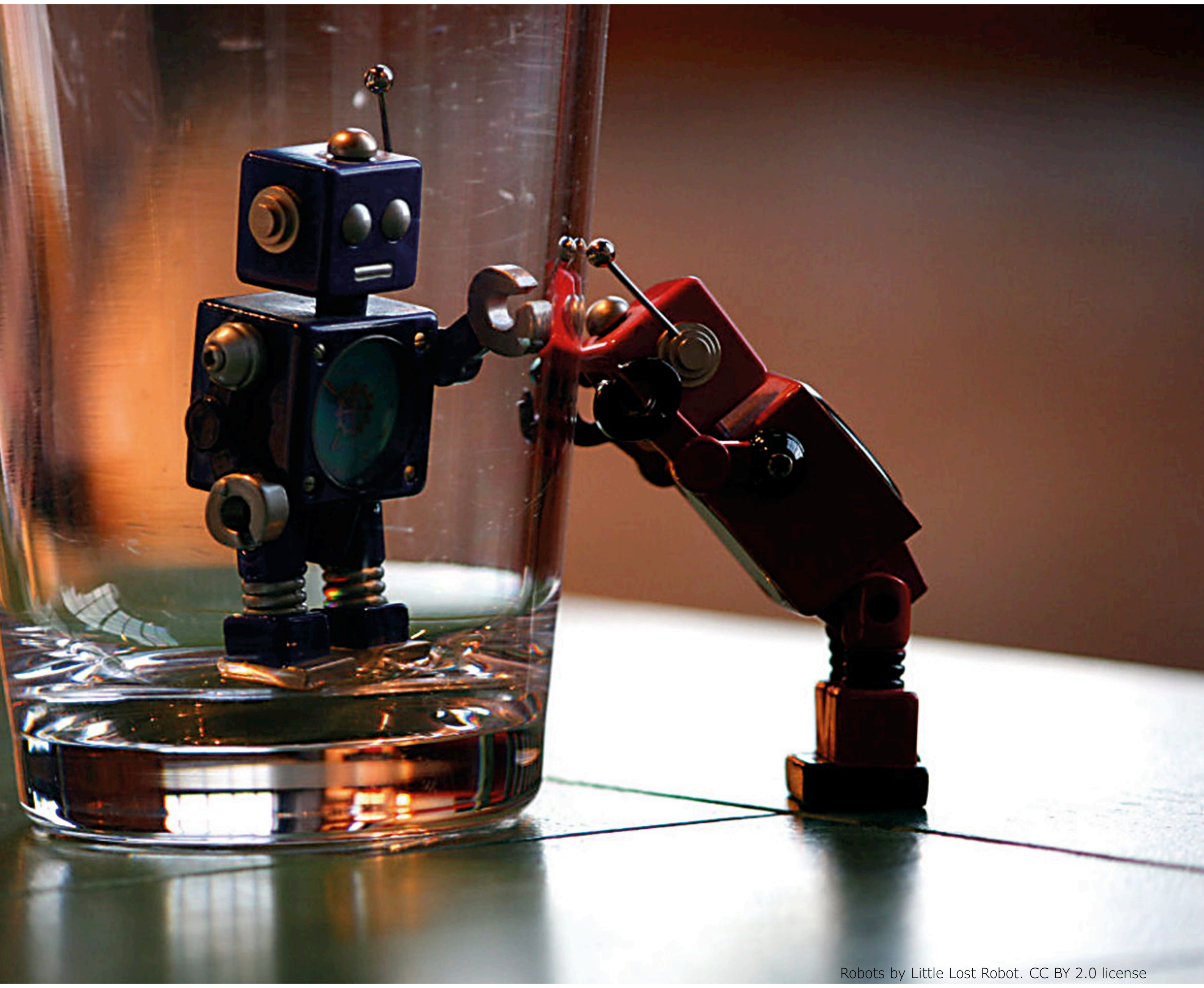
Next to that the EP urges to be mindful of the potential risks AI and robotics can pose to people and society as a whole. The development and use of robots should therefore always adhere to European values enshrined in the Treaty on European Union including: human dignity, equality, justice and equity, non-discrimination, informed consent and data protection.

Employment & education

The aspect of robots that has been most discussed in the public domain is the loss of jobs due to robotization. If many jobs disappear without being replaced we might have to rethink how societies are organized. The function of employment as a means to distribute wealth might no longer be practicable. In this context the EP asks the Commission to monitor this trend and to analyze the possible consequences for the viability of the social security systems of the Member States.
With regard to education it warns that there is a growing shortage of ICT professionals and that 90% of jobs require at least basic digital skills. It advises the EU to develop flexible training and education systems to help Europeans educate themselves to prepare for the increasing digitization and automation of the economy.

Liability

Another oft-discussed topic is that of liability. Who is responsible for the damage caused by a system if it is autonomous


Robots by Little Lost Robot. CC BY 2.0 license

and self-learning? The producer? The owner? The robot itself? If this issue is unresolved it could hamper innovation because both manufacturers and end-users lack legal certainty. On this matter the Parliament asks the Commission to propose a legislative instrument to address the question of liability.
And, finally the EP calls for the establishment of a European Agency for Robotics and AI which would serve as a center of expertise to help implement the aims mentioned above.

Public debate

The Commission is not obliged to follow the Parliament's proposal but it must state its reasons when it doesn't. At the time of writing the Commission has not yet responded. Should the Commission accept or reject the resolution?

Opponents say it is way too early for any robot law. The field is still emerging and we know too little about how robotics will develop. They also object to compounding AI and robotics which are two different fields. In fact, even within the field of robotics there are vast differences. A Roomba obviously needs less oversight than a care robot tending to the elderly. The most potent objection is that too much legislation will stifle innovation.
Proponents beg to differ. They argue that a common framework will actually promote innovation. First, because it will provide legal certainty for both manufacturers and end-users. And, second, because it will harmonize technological standards across the different Member States.

Whether we need a legislative framework for robotics right now is debatable. But it certainly is time to start thinking about it. Rather than have the next wave of technological progress wash over us, we need a public debate about what we want out of robotics and AI. How our society is shaped by these emerging technologies, should be decided in a democratic way. ⏮

(160334)

welcome in your

ONLINE STORE

EDITOR'S CHOICE

MiniPro TL866A Universal Programmer

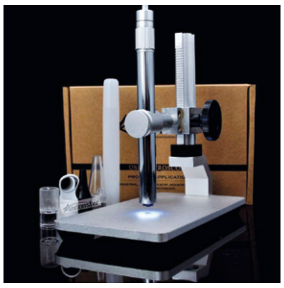
Most modern microcontrollers with flash memory support in-circuit programming. However, it's not been that long since every self-respecting electronics hobbyist wished they could afford a device for programming PROMs, EPROMs, GALs and microcontrollers. Preferably one that could handle a lot of different types. That used to be out of the question for most of us, but now you can have a universal programmer on your bench at a very affordable price in the form of the MiniPro TL866A. This programmer is very suitable for developing prototypes for hobby use. However, none of the programming algorithms is certified, so for the production of commercial products you should look for a professional certified programmer. Aside from that, the MiniPro TL866 is highly recommended, particularly considering the price, the features and the included accessories.

Luc Lemmens (Elektor Labs)

www.elektor.com/tl866a-universal-programmer

Elektor Bestsellers

1. Digital Microscope V160
   www.elektor.com/microscope-v160



2. Raspberry Pi 3 (Model B)
   www.elektor.com/rpi3
3. BBC micro:bit (Book)
   www.elektor.com/microbit-book
4. Sand Clock
   www.elektor.com/sandclock
5. DVD Elektor 2016
   www.elektor.com/dvd-elektor-2016
6. Mastering Microcontrollers Helped by Arduino
   www.elektor.com/mm3
7. BBC micro:bit (Board)
   www.elektor.com/microbit

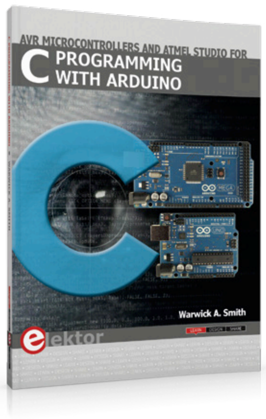
BBC micro:bit – 35 Touch Develop & MicroPython Projects

This book is about the use of the BBC micro:bit computer in practical projects. The BBC micro:bit computer can be programmed using several different programming languages. This book makes a brief introduction to the Touch Develop programming language and the MicroPython programming language. It then gives 35 example working and tested projects using these language. Complete program listings are given for each project.

member price: £19.95 • €22.46 • US $24

www.elektor.com/microbit-book

C Programming with Arduino


AVR MICROCONTROLLERS AND ATMEL STUDIO FOR
C PROGRAMMING WITH ARDUINO
Warwick A. Smith
elektor

Technology is constantly changing. New microcontrollers become available every year. The one thing that has stayed the same is the C programming language used to program these microcontrollers. If you would like to learn the C programming language to program microcontrollers, then this book is for you! Arduino is the hardware platform used to teach the C programming language and Atmel Studio is used as the development environment.

member price: £34.95 • €40.46 • US $43

www.elektor.com/c-with-arduino

SmartScope Maker Kit

The new SmartScope Maker Kit (available exclusively from Elektor) contains a version of the SmartScope that's specially prepared for FPGA development, with ready assembled internal headers and additional FPGA pins that permit reading and writing. The remainder of the kit contents, including 2 programmers and 2 probes, is also highly worthwhile. Use the SmartScope as an FPGA development platform with this special Maker kit!

member price: £229.95 • €269.96 • US $287

www.elektor.com/smartscope-kit

NEW!

GSM/GPRS Projects

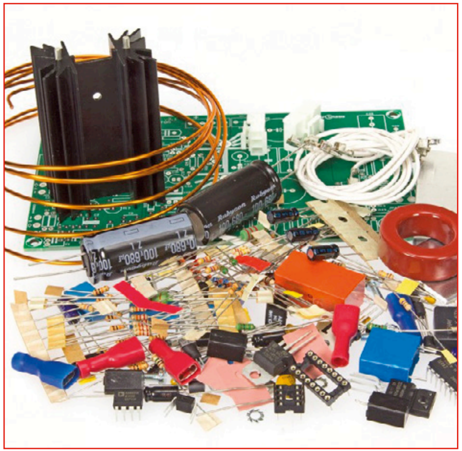
Based on PIC Microcontrollers and Arduino

Every mobile phone includes a GSM/GPRS modem which enables the phone to communicate with the external world. This book is aimed at people who may want to learn how to use the GSM/GPRS modems in microcontroller based projects. Two types of popular microcontroller families are considered in the book: PIC microcontrollers and the Arduino. The highly popular mid-performance PIC18F87J50 microcontroller is used in PIC based projects together with a GSM Click board. In addition, the SIM900 GSM/GPRS shield is used with the Arduino Uno projects. Both GSM and GPRS based projects are included in the book.

MEMBER PRICE: £21.95 • €24.95 • US $28

www.elektor.com/gsm-gprs-projects

D-Watt Amplifier

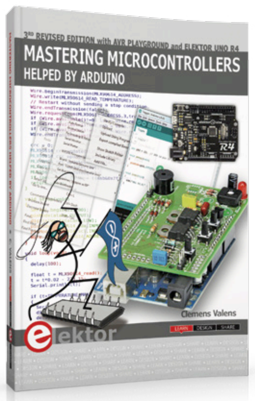


This high-performance amplifier is built around a digital audio driver IC and operating in class-D. This has the advantage that the amplifier can deliver a lot of output power (like 200 watts into 8 ohms) with very low heat dissipation. Please note that soldering is required. Case and power supply are not included in the kit.

member price: £84.95 • €98.96 • US $105

www.elektor.com/d-watt

Mastering Microcontrollers Helped by Arduino



This book will not only familiarize you with the world of Arduino but it will also teach you how to program microcontrollers in general. Having completed this fun and playful course, you will be able to program any microcontroller, tackling and mastering I/O, memory, interrupts, communication (serial, I²C, SPI, 1-wire, SMBus), A/D converter, and more. This third, extended and revised edition contains two new chapters.

member price: £32.95 • €38.66 • US $41

www.elektor.com/mm3

Sand Clock

This supercool gadget built around an Arduino Uno writes the time into a layer of sand. After an adjustable time the sand is flattened out by two vibration motors and everything begins all over again. This kit contains everything to build this sand clock by yourself: all mechanical parts, the motors, an Arduino Uno, a special RTC/driver shield, a power adapter and even a small bag with sand.

member price: £91.95 • €107.96 • US $115

www.elektor.com/sandclock

Hexadoku The Original Elektorized Sudoku

Traditionally, the last page of Elektor Magazine is reserved for our puzzle with an electronics slant: welcome to Hexadoku! Find the solution in the gray boxes, submit it to us by email, and you automatically enter the prize draw for one of five Elektor book vouchers.

The Hexadoku puzzle employs numbers in the hexadecimal range 0 through F. In the diagram composed of 16 × 16 boxes, enter numbers such that **all** hexadecimal numbers 0 through F (that's 0-9 and A-F) occur once only in each row, once in each column and in each of the 4×4 boxes (marked by the thicker black lines). A number of clues are given in the puzzle and these determine the start situation.

Correct entries received enter a prize draw. All you need to do is send us **the numbers in the gray boxes**.

Solve Hexadoku and win!

Correct solutions received from the entire Elektor readership automatically enter a prize draw for five Elektor Book Vouchers worth **$70.00 / £40.00 / €50.00 each**, which should encourage all Elektor readers to participate.

Participate!

Ultimately May 26, 2017, supply your name, street address and the solution (the numbers in the gray boxes) by email to: **hexadoku@elektor.com**

Prize Winners

The solution of Hexadoku in edition 3/2017 (March & April) is: **314AB**.

The €50 / £40 / $70 book vouchers have been awarded to: Sake van der Schaaf (The Netherlands), Eugene Stemple (USA), Michael Düren (Germany), Denis Moucharte (Belgium), and József Nagy (Hungary).

Congratulations everyone!

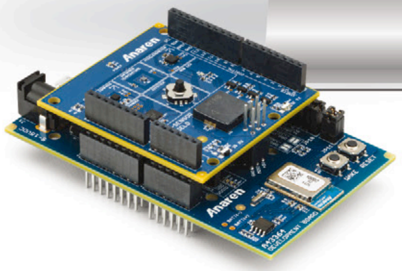
2	4		C	F		B	3	6	A	▨	▨	▨	▨	▨	
7	B					4	6		9			A			
5		6			A		1		C		2	4	8		
			A		8	9	E	7	D				3		
	7	5			D			3				B	4		2
		2				A			F		8				E
	F		D	E	3				B		1	5		0	
		3		1	0		5			9					
		7		B	E		8			C					
	5		8	9	1				2		6	C		3	
		C				F			3		4				8
	6	E			C			F				9	B		0
			5		4	1	C	0	6				2		
C		F			7		0		4		3	D	9		
6	3					8	D		1			0			
A	D		2	5		6	B	9	8						

B	9	8	0	5	6	4	A	D	C	E	1	F	3	7	2
5	A	7	D	1	F	C	8	0	2	3	6	E	9	4	B
C	E	1	2	B	9	7	3	F	4	8	A	D	5	6	0
F	4	3	6	D	E	0	2	9	5	7	B	C	1	A	8
1	7	B	C	9	A	2	D	5	3	4	F	0	E	8	6
E	D	5	A	F	0	8	7	1	6	C	9	2	4	B	3
2	F	4	3	C	5	B	6	7	8	0	E	9	A	1	D
0	6	9	8	E	3	1	4	A	B	D	2	5	C	F	7
3	0	C	F	2	7	E	B	6	9	A	D	4	8	5	1
A	5	D	4	8	1	9	0	3	E	F	7	6	B	2	C
6	8	E	1	3	C	D	5	B	0	2	4	A	7	9	F
7	B	2	9	6	4	A	F	8	1	5	C	3	D	0	E
4	C	F	B	7	D	3	1	2	A	9	0	8	6	E	5
8	2	A	7	0	B	5	E	4	D	6	3	1	F	C	9
9	3	6	5	A	2	F	C	E	7	1	8	B	0	D	4
D	1	0	E	4	8	6	9	C	F	B	5	7	2	3	A

The competition is not open to employees of Elektor International Media, its subsidiaries, licensees and/or associated publishing houses.

Take Off and Soar

With the Anaren Atmosphere 2.0 IoT Development environment and Atmosphere Cloud™ hosting, – *plus* – the new A43364 Wi-Fi development board, pilot your IoT project from concept to cloud connectivity at Mach speed.
No delays. No runaway costs.
First class all the way.



Scan the code to learn more and register for a chance to win a free Atmosphere MSDK!*
Only the Atmosphere 2.0 IoT Development enviroment with Atmosphere Cloud™ hosting gives you the fastest trajectory for developing connected, embedded Wi-Fi applications, complete with mobile apps and cloud support.
Think. Build. Connect.

ATMOSPHERE

www.anaren.com/air/win

* Must be 18 to participate. No purchase necessary.
North America only. Void where prohibited. Other restrictions apply.

Anaren 50 YEARS

Anaren®
What'll we think of next?®

reichelt elektronik

Technology connects.

- More than 45 years of experience
- 24-hour shipping
- More than 70,000 products

DISCOVER FLUKE MEASURING TECHNOLOGY

SPECIALLY PRICED!

FLUKE

Precise, user-friendly, safe and reliable

TRMS digital multimeter

- True RMS measurement of voltage and current, 1000 V / 10 A
- 0.1 % base accuracy
- LCD, 6000 count resolutions
- Manual and automatic area selection
- Frequency, capacitance and resistance measurement

SET INCLUDES POUCH & TEST LEADS

EN 61010-1 CAT III 1000 V

EN 61010-1 CAT IV 600 V

instead of 251.41 FLUKE 175 PROMO

SAVE 44 % 140.61

For precise measurement of load current, voltage or resistance:

True RMS clamp-on ammeter

- AC current measurement up to 400 A
- DC and AC voltage measurement up to 600 V
- TRMS measurement for precise results with non-linear signals
- Resistance measurement up to 40 kOhm
- Temperature, capacitance and frequency measurement

EN 61010-1 CAT IV 300 V

EN 61010-1 CAT III 600 V

SET INCLUDES HOLSTER & CASE

instead of 275.33 FLUKE 325 PROMO

SAVE 38 % 170.36

FLUKE

Voltage detector with the latest measurement and safety technology!

T150 VDE with LED & LCD display

- Continuity test & resistance measurement
- Phase sequence indicator for 3-phase systems
- Function for checking RCDs
- Integrated flashlight

EN 61010-1 CAT III 600 V

EN 61010-1 CAT IV 600 V

FLUKE T150 VDE

PRICE TIP 111.44

FLUKE

SUBSCRIBE TO NEWSLETTER NOW & WIN!

Every month we raffle off a technological highlight among all new subscribers to the newsletter!

TAKE PART NOW! ▶ http://rch.lt/Ay

Daily prices! Price as of: 21. 3. 2017 Prices in £ plus statutory VAT, plus shipping costs · reichelt elektronik, Elektronikring 1, 26452 Sande (Germany) Onlineshop languages:

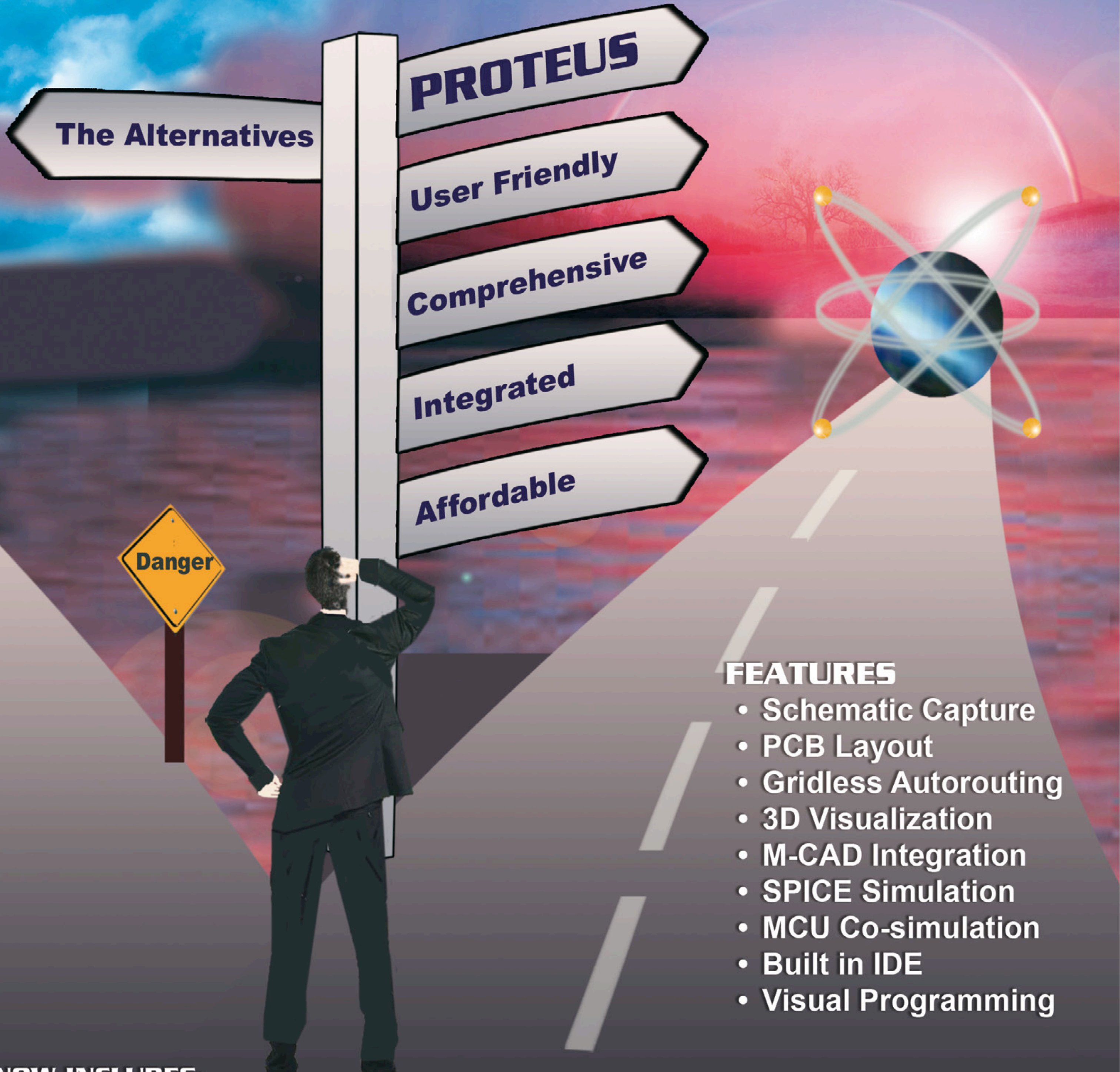
PAYMENT METHODS: VISA MasterCard PayPal

SHOP CONVENIENTLY ONLINE!

www.reichelt.co.uk

ORDER HOTLINE: +44 203 808 95 25

DO YOU WANT THE BEST ELECTRONICS DESIGN SOFTWARE



FEATURES
- Schematic Capture
- PCB Layout
- Gridless Autorouting
- 3D Visualization
- M-CAD Integration
- SPICE Simulation
- MCU Co-simulation
- Built in IDE
- Visual Programming

NOW INCLUDES:
Serpentine Routing, Layer Stackup Manager, and Assembly Variants.

labcenter Electronics

www.labcenter.com

(+44) 01756 753440