www.elektor-magazine.com

.magazine

June 2013

elektor

WiFi Remote Control

for LED ribbon strips, relays, actuators and motors

Wi-Fi Controller Board | **Recycle your ATX Power Supply**

Off to the EF Library | Lost Model Finder | Lithium-ion Battery Recycling Made Easy | **Taming the Beast (5)** | From BASIC to Python (2) | **LCR Meter Shootout** | **Konrad Zuse's Z1 through Z4**

US $9.00 - Canada $10.00

06

7 25274 24965 7

HuMANDATA LTD. www.hdl.co.jp/EL/ E-mail : s2@hdl.co.jp Fax : +81-72-620-2003

HUMANDATA® FPGA / CPLD Boards from JAPAN

SAVING COST=TIME with readily available FPGA boards

- Basic and simple features, single power supply operation
- Quality and reliability is provided by years of sales
- Same board size and connector layout – ACM/XCM series
- All stocked items are ready to be shipped immediately
- Over 100 varieties of FPGA/CPLD boards are available
- Customizing speed grade and/or any features are possible
- Free download technical documents before purchasing
- High quality and highly reliable FPGA/CPLD boards from Japan
- Almost all products are RoHS compliance

5" LCD Touch Panel Module

5 inch TFT full color LCD display with WVGA(800x480) resolution resistive touch panel

NEW **UTL-021**

- 3.3 V single power supply operation
- Piezo buzzer to beep
- Useful plastic bezel is included to assemble
- LTM-compatible pin assignment RoHS

ALTERA FPGA Board

Cyclone IV E F780 FPGA board

ACM-204 series NEW

Cyclone IV E | SDRAM

EP4CE30F29C8N
EP4CE40F29C8N
EP4CE115F29C8N
Credit card size (86 x 54 mm)

RoHS compliant

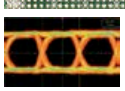
Arria II GX F572 FPGA board

ACM-025 series

Arria II GX | DDR2 | SIF40

EP2AGX45DF25C6N
EP2AGX65DF25C6N
EP2AGX95DF25C6N
EP2AGX125DF25C6N
Credit card size (86 x 54 mm)

RoHS compliant



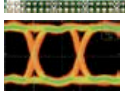
CycloneIV GX F484 FPGA board

ACM-024 series

Cyclone IV GX | DDR2 | SIF40

EP4CGX50CF23C8N
EP4CGX75CF23C8N
EP4CGX110CF23C8N
EP4CGX150CF23C7N
Credit card size (86 x 54 mm)

RoHS compliant



XILINX FPGA Board

Spartan-6 FGG484 FPGA board

XCM-018/018Z series

Spartan-6 | MRAM | DDR2

XC6SLX45-2FGG484C
XC6SLX75-2FGG484C
XC6SLX100-2FGG484C
XC6SLX150-2FGG484C
Credit card size (86 x 54 mm)

RoHS compliant

Virtex-5 FFG676 FPGA board

XCM-109 series

Virtex-5 | SDRAM

XC5VLX30-1FFG676C
XC5VLX50-1FFG676C
XC5VLX85-1FFG676C
XC5VLX110-1FFG676C
Compact size (43 x 54 mm)

RoHS compliant

Spartan-6 FGG676 FPGA board

XCM-206 series

Spartan-6 | MRAM | DDR2

XC6SLX100-2FGG676C
XC6SLX150-2FGG676C
Credit card size (86 x 54 mm)

RoHS compliant

Spartan-3A FTG256 FPGA board

XCM-305 series

Spartan-3A | MRAM

XC3S700A-4FTG256C
XC3S1400A-4FTG256C
Compact size (54 x 53 mm)

RoHS compliant

USB- FPGA Board

Cyclone IV USB-FPGA Board

EDA-301 NEW

Cyclone IV E | USB Config. | USB Comm. | HI-SPEED

EP4CE15F17C8N
Compact size (54 x 53 mm)

RoHS compliant

Spartan-6 USB-FPGA board

EDX-301 NEW

Spartan-6 | USB Config. | USB Comm. | HI-SPEED

XC6SLX16-2CSG225C
Compact size (54 x 53 mm)

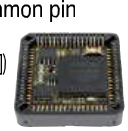
RoHS compliant

FPGA/CPLD Stamp Module PLCC68 Series

Easy and Quickly Mountable Module

FPGA Module IC socket mountable

- 50 I/Os (External clock inputs are available)
- 3.3V single power supply operation (Voltage converters for auxiliary power supply are built-in)
- Separated supply-inputs: Core, I/O drivers
- JTAG signal
- All PLCC68 series have common pin assignment



- Very small size (25.3 x 25.3 [mm])
- RoHS compliance
- MADE IN JAPAN

XILINX PLCC68 Series

Spartan-6 PLCC68 FPGA Module

XP68-03

Spartan-6 | PLCC 68

XC6SLX45-2CSG324C
3.3V single power supply operation
On-board oscillator, 50MHz

RoHS compliant

ALTERA PLCC68 Series

Cyclone III PLCC68 FPGA Module

AP68-04

NEW Cyclone III | PLCC 68

EP3C25U256C8N
3.3V single power supply operation
On-board oscillator, 50MHz

RoHS compliant

Universal Board (Type2)

ZKB-106 RoHS

- One for general power(3.3V 3A max) and the Two variable outputs for Vccio(0.8V to 3.3, 3A max)
- For ACM/XCM-2 series FPGA boards
- Power Switch and LED
- Power input:DC5V/2.1[mm] Jack/ Terminal Block (option)
- Board size :156x184 [mm]
- 4 Layers PCB, Thru-hole

See all our products, A/D D/A conversion board, boards with USB chip from FTDI and accessories at : **www.hdl.co.jp/EL/**

Join the Elektor Community

Take out a GOLD Membership now!

elektor GOLD membership

Your GOLD Membership contains:

- 8 Regular editions of Elektor magazine in print and digital
- 2 Jumbo editions of Elektor magazine in print and digital (January/February and July/August double issues)
- Elektor annual DVD-ROM
- A minimum of 10% DISCOUNT on all products in Elektor.STORE
- Direct access to Elektor.LABS
- Direct access to Elektor.MAGAZINE; our online archive for members
- Elektor.POST sent to your email account (incl. 25 extra projects per year)
- An Elektor Binder to store these 25 extra projects
- Exclusive GOLD Membership card

ALSO AVAILABLE:

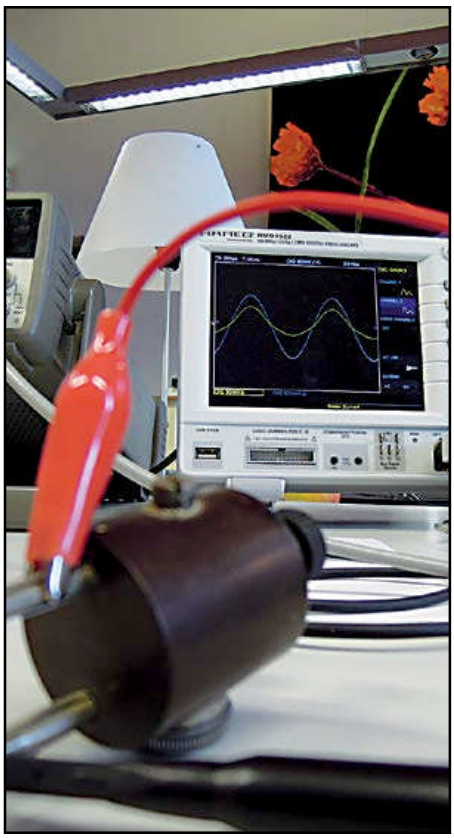
The all-paperless GREEN Membership, which delivers all products and services, including Elektor.MAGAZINE, online only.

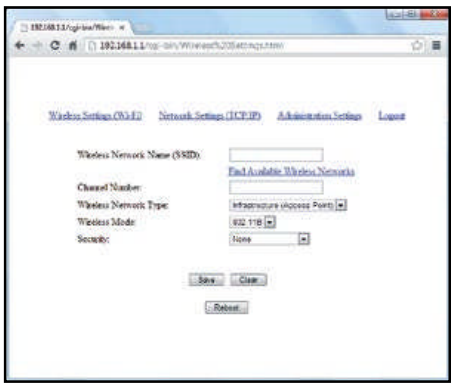
elektor GREEN membership

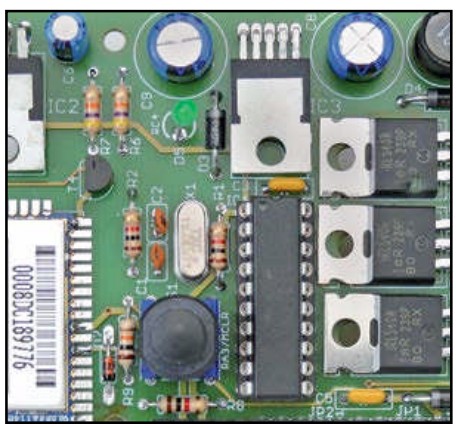
Take out your Membership now at www.elektor.com/members

Contents

elektor magazine



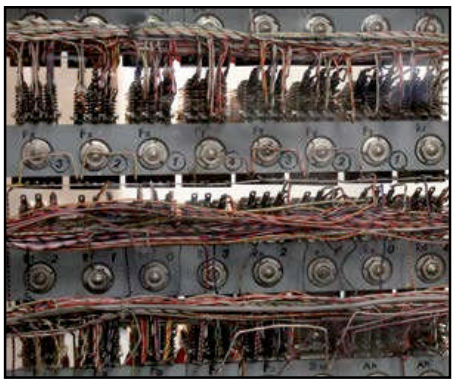


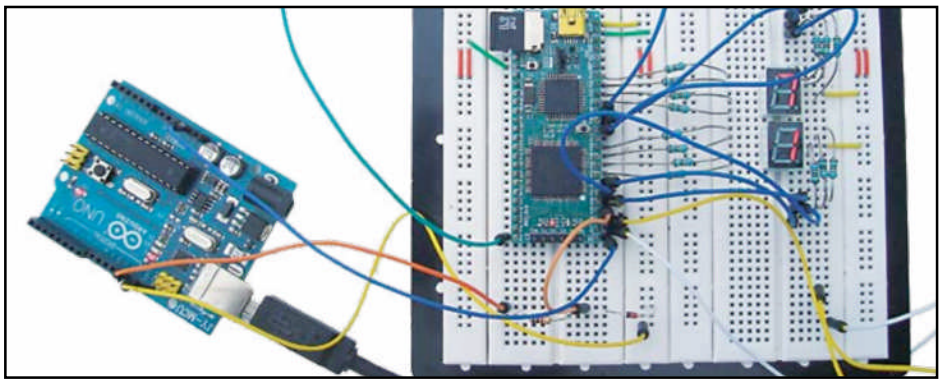


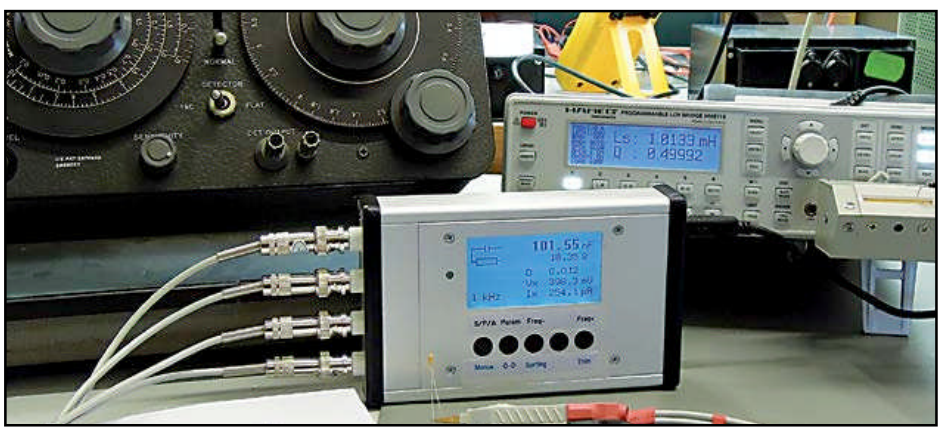
Community

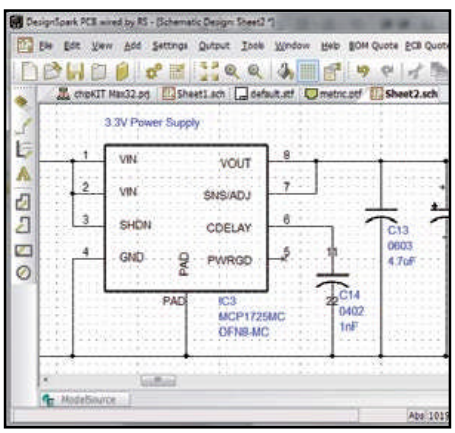
DesignSpark

Projects









is usually done with a hardware description language. An advantage of the latter approach is that complex functions are often easier to express in algorithms than in schematics. Accordingly, in this installment we guide you through the process of programming an FPGA application.

Labs

Industry

Magazine

No. 54, ~June 2013
ISSN 1947-3753

Elektor Magazine is published 10 times a year including double issues in January/February and July/August at $80 per year, Canada add $15 per year; by

Elektor International Media LLC
111 Founders Plaza, Suite 300
East Hartford, CT 06108.

Phone: 860.289.0800
Fax: 860.461.0450
www.elektor.com

Elektor is also published in French, Spanish, German and Dutch. Together with franchised editions the magazine is on circulation in more than 50 countries.

Memberships:
Elektor USA
P.O. Box 462228
Escondido, CA 92046.

Phone: 800-269-6301
E-mail: elektor@pcspublink.com
Internet: www.elektor.com

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

Advertising:
Strategic Media Marketing
Peter Wostrel
2 Main Street
Gloucester MA 01930.

Phone: 978-281-7708,
Fax: 978-281-7706
E-mail: peter@smmarketing.us
Advertising rates and terms available on request.

A Case for Boards

Looking at our readers' main interests, PCB design and production rank pretty high. Elektor PCBs are famous the world over not just for their quality, but also their consistent look and feel. How did that come about?
Here at Elektor, the change from manual artwork design using masking tape and photographic reproduction techniques to a 100 % PC-driven process was gradual, and took place in the early 1990s. The use of a PC to draw a schematic and then run a PCB design program was not forced or even suggested by the publishers at the time. Back then, some of the younger lab designers boldly set out to discover the advantages of the PC route, eventually supplying files instead of drawings to their colleagues in the PCB design department. Others stuck to pencil, paper and rubber with equally good results particularly in RF and space critical designs. No matter how the final artwork got produced, Elektor never actually mass-produced their circuit boards—this was always farmed out to PCB manufacturers. We did, however, handle the storage and packaging of what must have amounted to hundreds of thousands of those blue and green boards. Also, to this day Elektor Labs have their own PCB etching and drilling facilities. The equipment is used to make prototypes and one-offs of any board, single or double sided, TH or SMD.
I do recall the excitement in the lab and editorial offices about 20 years ago when a parcel arrived containing 500 or so boards for a recently published project. At last, the proud designer was able to see the fruit of his design efforts. More importantly however, readers all over the world were able to construct circuits on superbly produced circuit boards with a component overlay and silk screen finish! Today, there is still the satisfaction not only of publishing these wonderful designs and getting response from you, but also of holding a perfectly machined printed circuit board with an Elektor production number printed to aid identification.

Jan Buiting, Managing Editor

The Team

Managing Editor:	Jan Buiting
Publisher:	Hugo Van haecke
Membership Manager:	Shannon Barraclough
International Editorial Staff:	Harry Baggen, Thijs Beckers, Eduardo Corral, Wisse Hettinga, Denis Meyer, Jens Nickel, Clemens Valens
Laboratory Staff:	Ton Giesberts, Luc Lemmens, Tim Uiterwijk, Clemens Valens, Jan Visser
Graphic Design & Prepress:	Giel Dols, Jeanine Opreij, Mart Schroijen
Online Manager:	Daniëlle Mertens
Managing Director:	Don Akkermans

USA
Hugo Van haecke
+1 860-875-2199
h.vanhaecke@elektor.com

United Kingdom
Wisse Hettinga
+31 46 4389428
w.hettinga@elektor.com

Germany
Ferdinand te Walvaart
+49 241 88 909-17
f.tewalvaart@elektor.de

France
Denis Meyer
+31 46 4389435
d.meyer@elektor.fr

Netherlands
Harry Baggen
+31 46 4389429
h.baggen@elektor.nl

Spain
Eduardo Corral
+34 91 101 93 95
e.corral@elektor.es

Italy
Maurizio del Corso
+39 2.66504755
m.delcorso@inware.it

Sweden
Wisse Hettinga
+31 46 4389428
w.hettinga@elektor.com

Brazil
João Martins
+55 11 4195 0363
joao.martins@editorialbolina.com

Portugal
João Martins
+351 21413-1600
joao.martins@editorialbolina.com

India
Sunil D. Malekar
+91 9833168815
ts@elektor.in

Russia
Nataliya Melnikova
+7 (965) 395 33 36
Elektor.Russia@gmail.com

Turkey
Zeynep Köksal
+90 532 277 48 26
zkoksal@beti.com.tr

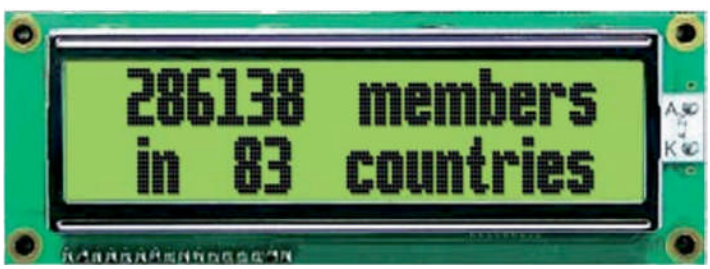
South Africa
Johan Dijk
+31 6 1589 4245
j.dijk@elektor.com

China
Cees Baay
+86 21 6445 2811
CeesBaay@gmail.com

Our network

elektor · VOICE COIL · CIRCUIT CELLAR · hifi.nl MUZIEK - FILM - LIFESTYLE · audioXPRESS · techthefuture

Connects you to



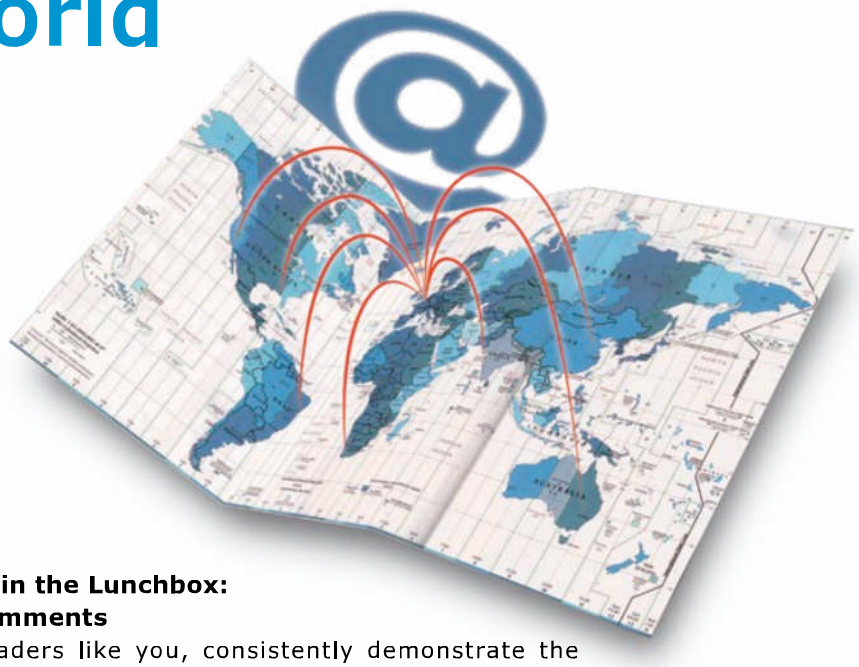
Supporting Companies

Not a supporting company yet?

Contact Peter Wostrel (peter@smmarketing.us, Phone 978-281-7708, Fax 978-281-7706) to reserve your own space for the next edition of our members' magazine

Elektor World

Compiled by
Wisse Hettinga



The Tube in the Lunchbox: reader comments

Elektor readers like you, consistently demonstrate the true spirit behind the magazine, enhancing and extending the editorial content with your own investigations and experiments. Component related articles remain the most popular and in last April's *Elektor World* we showed the 'Tube in the Lunchbox' and asked for more details.
Among others, Christopher Kessler from Germany and Jan Swenker from The Netherlands dug out more information on this. Christopher found the specs of this tube in the *"Valvo Fotovervielfacher 1978-79".* In total he found 6 pages of information on the device and figured that its current price tag was around 30 dollars. Jan's information confirms the date of origin: 1978. He pointed us also to the *Hamamatsu Photomultiplier Tubes* book from 1986 (you can find the PDF on the internet). The book has the specs of a replacement tube, the model R1450... It is time to get our hands dirty getting this tube to work!
Many thanks to Jan and Christopher

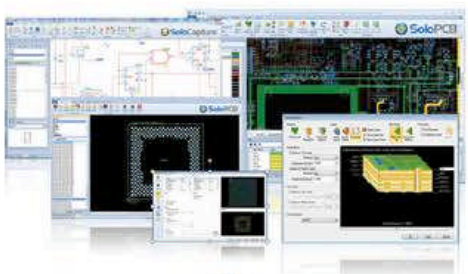
Getting control

"In engineering, control system theory focuses on how to manipulate a system's inputs to change its behavior. DIYers are familiar with many simple, open-loop control systems — such as those in basic stepper motors. But today, project kits for everything from robotic sumo cars to auto-piloted model aircraft demand DIYers learn more about the theory behind complex, closed-loop control systems", Brian Douglas says in his "Tech the Future" essay in *Circuit Cellar* magazine's June 2013 edition. "DIYers aren't going back to the classroom, they're going online — educating themselves on websites and turning to open-source software and hardware". Douglas should know. The Seattle-based control systems engineer has a YouTube site dedicated to fostering a practical understanding of control system theory, www.youtube.com/user/ControlLectures. Douglas is one of the essayists featured in CC's recurring "Tech The Future" section, which focuses on developing trends and breakthroughs in technology. Visit circuitcellar.com/category/tech-the-future to read more of Douglas's essay, and others.

WE MAKE PCB DESIGN AND MANUFACTURING AS EASY AS...

Introducing FabStream...

An integrated approach that combines FREE PCB design software with industry-leading PCB manufacturers



FREE PCB Design Software

Our Solo PCB Design® software is a free, full-featured schematic capture, layout and autorouting tool suite. It's customized specifically for each of our manufacturing partners to automate design setup right from the start and design within your chosen manufacturer's capabilities to minimize defect or delay.

Your Choice of PCB Manufacturing Partners

We've created a global network of industry-leading PCB manufacturers to give you a variety of manufacturing options. Choose your manufacturer based on your design specs, geographic location, budget and delivery needs, then use the integrated SoloPCB ordering wizard to automatically price your PCB order and extract and upload all the necessary manufacturing data directly to your chosen manufacturer 24/7/365.

Get Started Now at FabStream.com

FabStream™
A DownStream Technologies Solution

Day 1 – Make it just the way you want

By **Neil Greunding**

The good folks at RS Components have just released version 5.0 of DesignSpark PCB and the first thing I like to do with a new tool is to get it configured just the way I like it. DesignSpark lets you configure everything on a per file basis or globally.

Today I will walk you through how to make global setting changes using DesignSpark technology files so that you can make DesignSpark work just the way you want.

Getting started

But before we start configuring DesignSpark it's important to know that DesignSpark uses styles to specify the formatting rules for design primitives like shapes, text and tracks. Each style has a name to make it easy to reference, just like in a word processor. I normally try to give styles meaningful names so that you can know what the style is without checking its properties directly. For example, having a style named "Via" is ok if you only have one via but calling it "Via (0.45 mm x 0.95 mm)" makes it immediately obvious that the via has a 0.45 mm drill and a 0.95 mm copper pad. You are free to add custom styles to the technology files but this discussion we will be focusing on the system default styles like "[Symbol Names]" which is the style name used for reference designators and component names.

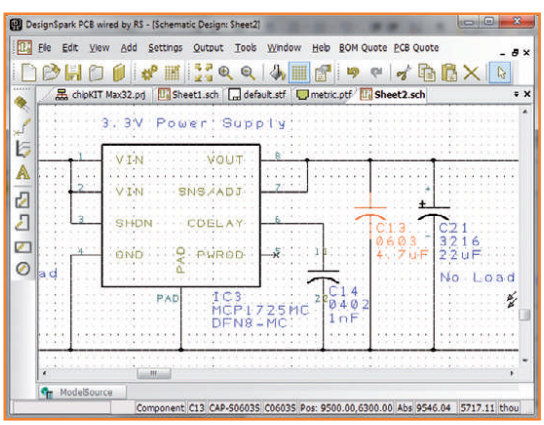


Figure 1.
Schematic with default parameters.

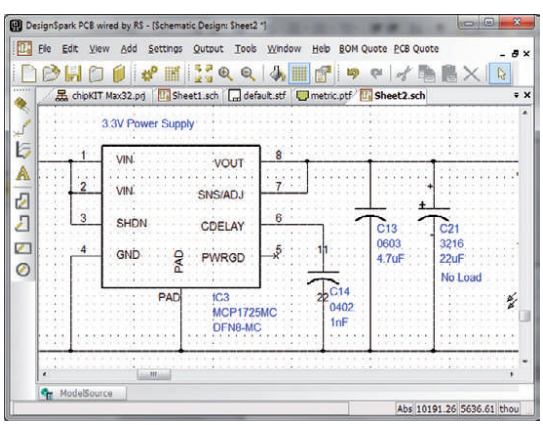


Figure 2.
Default Fonts set to Arial.

Also, don't forget to double check the DesignSpark technology file path which can be changed on the General tab in the Settings->Preferences menu. In my installation I had to change it to C:\Users\Public\Documents\DesignSpark PCB 5.0\Technology. If this path is incorrect DesignSpark won't find any technology files automatically which makes it harder to use. You can check if the directory path is correct if it contains .ptf and .stf files.

Schematic technology files

Schematic technology files are where you can configure:

- The line styles used for terminal and junction connections
- Predefined text styles used in the schematic (font, size, etc.)
- How various line elements are drawn (solid, dashed, etc.)
- How the connection line elements are drawn

(solid, width, etc.)
- Any predefined electrical nets, although I would do this in the schematic instead
- Any predefined electrical net classes (ground, power, etc.)
- The colors used to draw various elements

You can change these parameters from the Settings->Design Technology menu and in the View->Colors menu.
You can see what the default parameters look like by loading an example project. The chipKit Max32 project schematics should appear as shown in **Figure 1**.

I personally find that the stroke fonts look old fashioned and so I like to change them to Arial since it's a standard true type font. After playing around for a few minutes this is what I came up with **Figure 2**.

So how do we implement this in the schematic technology file? The first thing you need to do is open the default.stf schematic technology file which is normally located in C:\Users\Public\Documents\DesignSpark PCB 5.0\Technology which will open as a blank schematic document. Now open the Settings->Design Technology menu and change the Net Names, Pin Names, Pin Numbers and Symbol Names text styles to use an Arial font with a height of 80. I also changed the Normal text style to Arial as well but with a size of 120. I then went into the View->Colors menu and changed the Pin Names and Pin Numbers fields to black.
Once you've made all of your changes, save the technology file so you can use it for new schematic pages by selecting "default.stf" in the new document menu window

PCB technology files

PCB technology files are one of the best DesignSpark features because they let you combine all of your basic design rules and layer stackups into files that are easily reused. For example, I have a file for simple low cost 2-layer boards and another for 4-layer boards. I can then choose which file I want to use when creating a new PCB document. This feature isn't common with other PCB design packages.

PCB technology files let you configure:
- The design units (mm, mils, etc) and the resolution.
- The design grids, especially the working grid
- The layer stackup and colors
- The track spacing and clearance rules
- The pad and track styles, although you would usually specify the pad styles in your library components and only specify the default track styles in the technology file
- The default net classes, but I usually specify them in the schematic instead
- Autorouter and autoplacer rules
- Basic design elements like board shapes, mounting holes, etc.

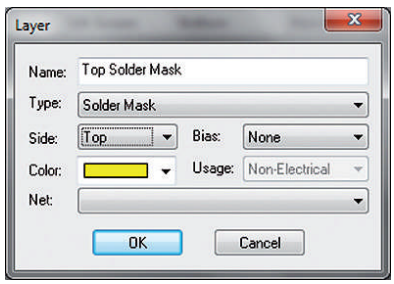


Figure 3.
Configuring the layers.

Let's see how this works by working through an example for a simple 2-layer board. The first step is copy and existing technology file so we don't have to start from nothing, so let's start with C:\Users\Public\Documents\DesignSpark PCB 5.0\Technology\metric.ptf to and save it with a new name like my2layer.ptf. The first thing to modify is the design units in Settings->Units menu. I always use mm with a precision of 4 decimal places, but if you prefer imperial units choose in or mil instead. You can then set up your preferred design grids in the Settings->Grids menu.

Configuring the layers in done on the Layers tab in the Settings->Design Technology menu. By default the metric technology file has a Top Silkscreen, Top Copper, Documentation, Bottom Copper and Bottom Silkscreen layers already defined. Since all my boards are surface mount, I added a Top Paste, Top Solder Mask, Bottom Solder Mask and a Bottom Paste layer. You can add layers by clicking on the Add button and entering the layer parameters as shown in the example of the Top Solder Mask layer in **Figure 3**.

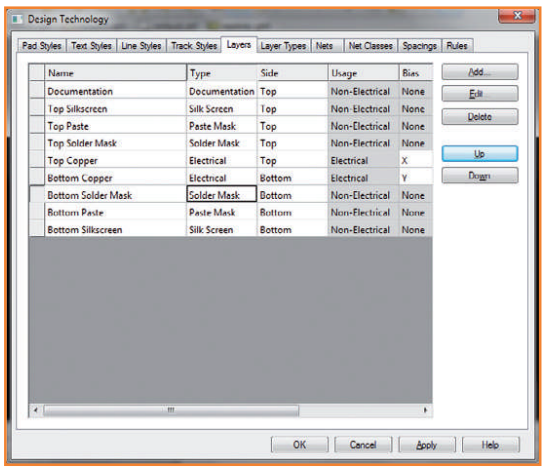


Figure 4.
Configuring the layers.

Once you've added all the layers you want, you can order them properly in the layer stackup by moving them up and down in the Layers window. When you're done you should see something like **Figure 4**.

Next, let's configure the spacing rules by clicking on the Spacings tab. Here you will see a matrix of all the spacing rules between the different object types. For a basic 2-layer board with 10-mil tracks and 10-mil spacing, the rules could look like in **Figure 5**.

Also don't forget to click on the Rules tab. The important parameters are the minimum annular ring and the component spacing. After that it's time to set the default track widths in the Track Styles tab. For a 10 mil/10 mil board I would set the minimum and normal signal track widths to 0.25 mm. The power track widths can be anything you want for the normal width, but I recommend making the minimum 0.25 mm so that you can route the power traces onto the component pads. The final step is to edit the via styles in the Pad Styles tab. For a basic technology 2-layer board I would use a via with a 0.45 mm drill with a 0.95 mm pad. You can also define other via styles if you like to use a variety of different via sizes. At this point you have a set of basic design rules and constraints for a basic 2-layer board that can be easily reused with future 2-layer boards. When you create a new PCB design, select the appropriate technology file when prompted in the PCB Creation Wizard.

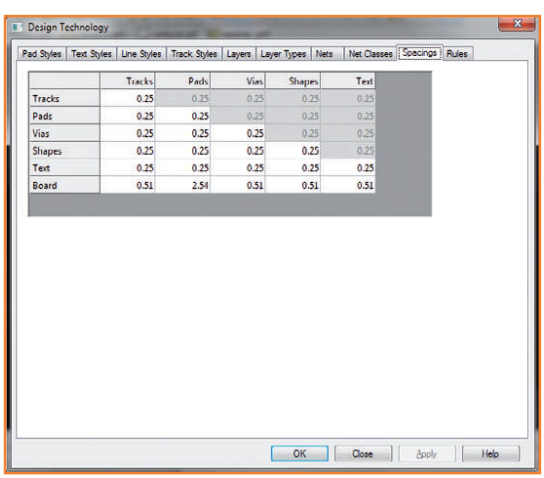


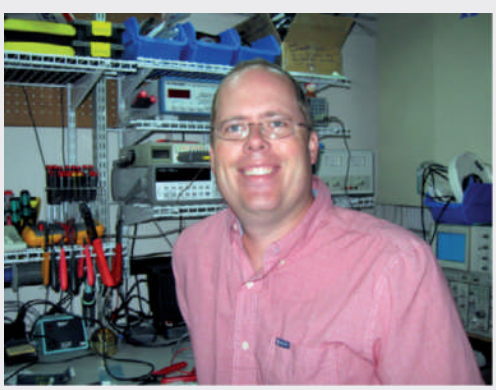
Figure 5.
Design rules.

Conclusion

Now that we have configured DesignSpark's default parameters the next steps are to configure DesignSpark's libraries and to create some documentation templates. Fortunately DesignSpark comes with a large set of libraries which makes getting started much easier.

(130172)

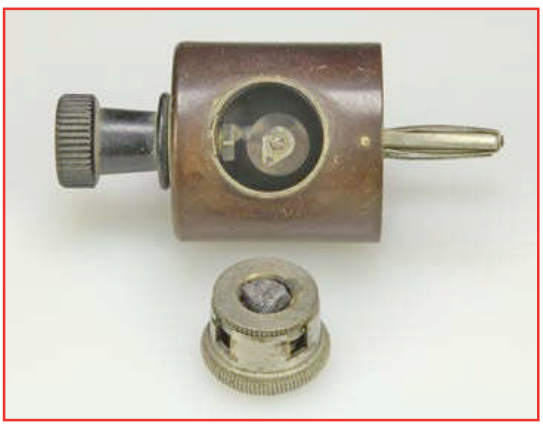
Hi, I'm Neil Gruending and I have used numerous different PCB CAD packages as an electronics design engineer over the years. I'm pretty particular about my tools and I like to learn how to maximize my productivity with them whenever possible. I also enjoy sharing what I've learned on my website at www.gruending.net and on Twitter as @ngruending.



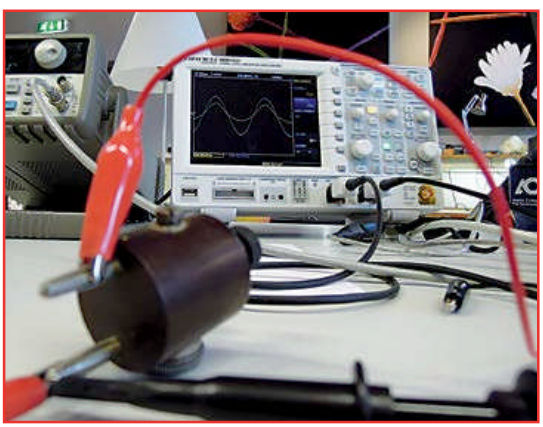
The Cat's Whisker

Wisse Hettinga
(Elektor)

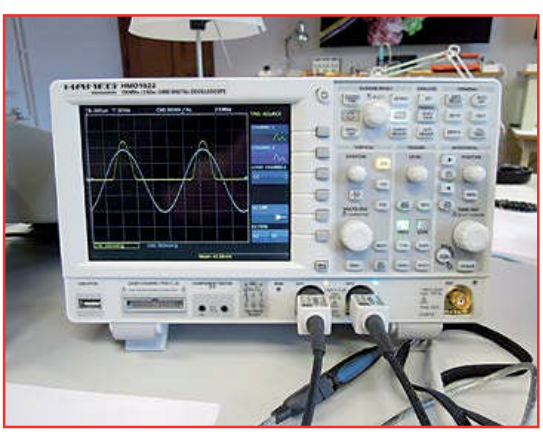
Usually when I start writing an article, for inspiration I look at what's around on my desk. For this edition of *Weird World of Components* a Crystal Diode type FRIHO D.R.P. caught my eye. It is mentioned in an old Radio Bauer Katalog from 1926 — we are talking old stuff here — and it is one of the main components of old crystal radio systems.



1



2



3

To be honest, I have no idea how or when I acquired this component, but in my early years I used to spend a lot of time browsing stuff in a renowned shop called Quakkelstein in Vlaardingen, The Netherlands. Most likely this was where I picked it up and since then it had been living on my desk or in my drawer. Now this component takes us back to the early days of Radio. It is an automated 'cat's whisker' diode. Turning the knob changes the position of a small wire (the whisker) on the galena crystal, allowing you to find the best spot for the diode to be doing what a diode should do: pass current in one direction only. This is what listening to the Radio encompassed those days. To get a decent signal you needed to twist and tweak the inductor, the capacitor and the rectifier diode.

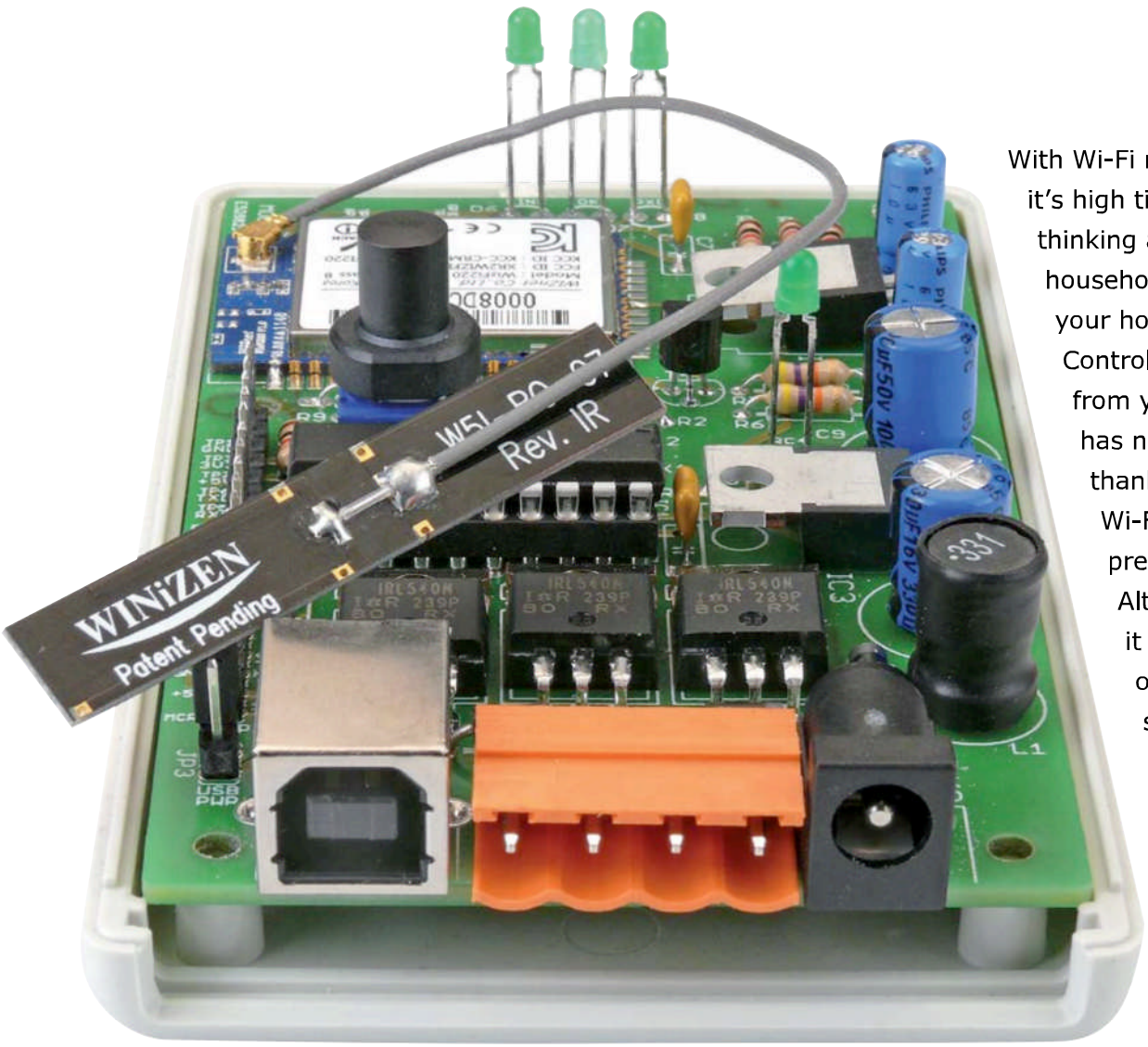
But how good where these diodes compared to what we have today? To measure the characteristics of a diode is really simple. Basically, if you want to do a decent job getting the *I*-*V* characteristic, all you need is a power supply, a voltage meter and an ammeter. Getting this diode to work properly is challenging to say the least. You need to fiddle around a bit to find the right spot where the diode effect peaks. I started out trying to find that spot with an ohmmeter, but soon I had to conclude that wasn't going to work. The measurement results were all over the place. I was ready to give up when I decided to give it a final shot and bring in some more equipment. An HP generator, a Hameg HMO1522 oscilloscope and a 330-ohm resistor finally gave me some promising results. In **Figure 2** the oscilloscope displays a very fuzzy signal, but with some imagination you can see the FRIHO cutting of the sinusoidal input signal. **Figure 3** shows the characteristic of a present-day diode in the same measurement setup.

Looking at the results I am amazed a component like the FRIHO D.R.P. actually worked! Today a huge variety of diodes exists for all kinds of applications. Small signal diodes, zener diodes, varicap diodes or varactors, tunnel diodes, and all of them are available in a variety of packages. It is interesting to realize they all relate to this very old Cat's Whisker diode in some way.

(130169)

Wi-Fi Controller Board

Control RGB LED strips, motors, relays & stuff, but no wires



With Wi-Fi modules pervasive it's high time to start thinking about adding household appliances to your home Wi-Fi network. Controlling your home from your mobile phone has never been easier thanks to the universal Wi-Fi controller board presented here. Although we used it to set the color of an RGB LED strip, you can use the board for a plethora of other applications.

By **Clemens Valens**
(Elektor.Labs)

The project you are about to encounter took a while to complete. It all started a year ago with a Home Automation system [1] originally developed by two trainees at Elektor Labs, Koen and Jesper. One part of their system was an RGB LED strip controlled over Ethernet, i.e. with a cable. The idea was nice, but I felt that a wireless connection would be a more appealing solution. Also, I wanted a web browser based application, compatible with most browsers, and that used sliders to control the color of the strip. So Koen set out to develop the lot, but unfortunately his trainee period ended before he could finish the project and he went back to college. Before he left Elektor, Koen had explained to me the problems he had run into, some of which still needed

addressing. I then took it on me to finalize the project, and so it spent the next few months on my desk gathering dust! When I finally found the time and energy to dive into it, I failed to recall most of what Koen had told me, and basically I had to start all over.

Koen's circuit was essentially an ATmega328 AVR microcontroller (MCU) controlling three MOSFETs with PWM signals to regulate the intensity of the three colors. For the Wi-Fi connection a WizFi220 module from WIZnet was used that communicates with the MCU over a simple serial link. Since I was in possession of an Arduino shield sporting this very Wi-Fi module [2] and since an Arduino Uno board is based on an ATmega328, I decided to build my prototype as an Arduino application. All I needed was a second shield with the MOSFETs on it so that I could drive the LED strip. This was quickly slapped onto a piece of prototyping board (**Figure 1**).

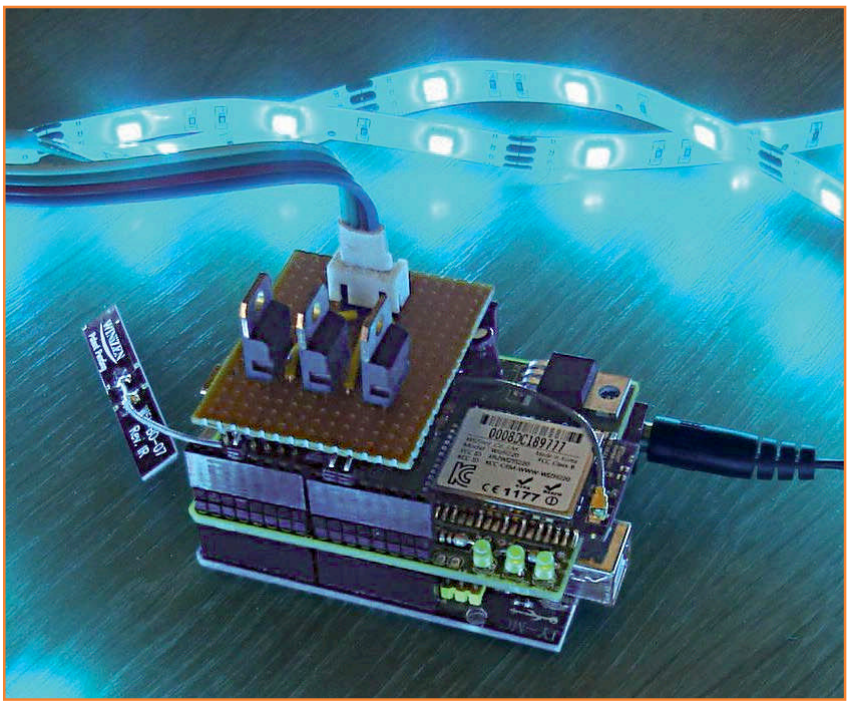


Figure 1.
The prototype based on an Arduino Uno, an Elektor Wi-Fi shield (120306) and a piece of prototyping board.

Then it was programming time. Parsing the HTTP commands received from the Wi-Fi module and sending back the required answers was not so complicated, but when I wanted to change the web page I ran into Koen's problems. First of all there were the sliders that control the LED strip's color from within a browser that I had so lightheartedly requested. As I then learned, there are no sliders in standard HTML. Searching the Internet I discovered that HTML 5 may be aware of sliders, but only very few browsers support it, so HTML 5 was not an option. Koen had solved this by using the JavaScript libraries JQuery and JQuery-UI [3]. These are online libraries for web pages to use to implement all kinds of nifty controls and other functions. Some inconveniences of these libraries are that they are online, meaning that you need an Internet connection to use them, and also that they are too large to put them in the MCU's program memory. Since I had no other solution, I decided to stick to the use of the online JavaScript libraries.

Looking through Koen's web page code I suddenly remembered the problem he had mentioned before he left: for some reason the Wi-Fi module closed the connection after receiving a command, disallowing the browser to send more commands. Consequently you could change the color of the LED strip just once, unless you rebooted the module. Koen had found a workaround using a complicated JavaScript that made the browser change its communication port before it would send a command. This workaround also involved using the HTTP GET command to send the color data, whereas this command is actually meant to get data from a server (who would have guessed that?). To send data you are supposed to use the POST (or PUT) command.

A much more elegant solution however is to add the line "Connection: close" to the server's reply to a GET command. Now I could delete the workaround script, making the web page much smaller, and I could switch to using the POST command, which, I felt, was more appropriate. This is what the POST command looks like when it is send by Firefox (the color data is at the end):

```
POST / HTTP/1.1
Host: 192.168.2.15
User-Agent: Mozilla/5.0 (Windows NT 6.1;
WOW64; rv:17.0) Gecko/17.0 Firefox/17.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Content-Type: application/x-www-form-
urlencoded; charset=UTF-8
Content-Length: 23
```

```
Origin: null
Pragma: no-cache
Cache-Control: no-cache

red=79&green=10&blue=20
```

The next step was to make the web page look pretty on mobile devices. On my smartphone the sliders were shown very small, making it impossible to move them with any precision. The solution was to add the `viewport` meta tag to the header of the web page, like this:

```
<meta name='viewport'
content='width=device-width, user-
scalable=no' />
```

With this line added to the HTML code of the web page it now nicely filled the screen of my Android phone while looking good on a PC as well. (On an iPad it occupied only about a quarter of the screen and I have not tried to improve that.)
Meanwhile I had also optimized the size of the rest of the program and now my goal was to make it all fit in as little memory as possible. An important improvement came from compressing the web page. You can do this with gzip and then add the line "`Content-encoding: gzip`" to the server's reply. Most, if not all, browsers will know how to handle zipped web pages. The disadvantage of doing this is that it is now a bit more complicated to change the web page; therefore you only do this when its design is finished.

At that point I had a fully working prototype based on Arduino compatible hardware with software that compiled within the Arduino IDE. If you want the details, please visit [4]. I could have stopped here, but I wanted better hardware; a stack of three PCBs for such a simple circuit just didn't feel right. Looking for an MCU with fewer pins I found that I could not stay on Atmel's AVR platform because the parts that would be suitable are difficult to get. However, I did happen to have a couple of Microchip PIC18F14K50 devices lying around. This is a 20-pin MCU with USB, which was interesting because that would allow for comfy PC-based configuration of the Wi-Fi module (see also [2]). Another interesting feature is the USB bootloader Microchip provides (for free), which enables easy firmware development without a special programmer. Unfortunately, this MCU has only one PWM channel, so I had to implement the three-channel PWM color control in software.
Porting the AVR code to the PIC should have been an easy and straight forward exercise, but it wasn't. Of course I had complicated things a bit by wanting to include a bootloader and USB functionality, but Microchip could have made it a bit easier. I was using my super-duper full-featured XC8 compiler that Microchip is raving about but found it impossible to get the USB code to compile, let alone to work. I got this code from the Microchip Application Libraries v2012-10-15 and only after many hours of fruitless fiddling with compiler settings and pragmas did I find a statement somewhere on the internet saying that XC8 is not (yet) compatible with the Microchip Application Libraries. Duh! I needed another compiler. Again internet came to the rescue and within fifteen minutes I had an official full-featured C18 compiler up and running without shelling out a dime. Another fifteen minutes more and I had the USB serial port example up and running.
Now it was time to add my Wi-Fi code — which went pretty smoothly except for fitting all the data in the PIC's segmented RAM — and try it out. And of course it worked. Not. Actually, it worked a bit. I could change the color of the strip once or twice, but then Wi-Fi communication would stop. When I controlled the strip over the USB serial port, it all worked fine. So, back to the debugging table where I discovered that the USB interrupt seemed to hold up the serial port interrupt, causing data loss. By this time I was pretty fed up with it all and decided to simply disable the USB port in Wi-Fi mode instead of going to the bottom of the problem. (I suggest you give it a try if it really interests you. Please let me know if you find the solution.)

Building it

Now my second — PIC-based — prototype was functioning as intended with the parts I wanted to use (**Figure 2**); the time to design a nice PCB had finally come. Since the goal was to control an LED strip, I decided to build it into a small, discreet enclosure. The PCB is designed in such a way that the enclosure requires minimal tooling, especially when you use the blue transparent version that allows you to view the status LEDs inside. In that case no drilling is needed at all; you only have to trim some standoffs inside. Even with a 20-pin MCU several pins remained unused so I added some extra functionality to the board. In combination with the bootloader

this made the board into a much more versatile platform that you can use for other applications as well. It can be used as:

- Wi-Fi RGB LED (strip) controller;
- three-channel controller to switch relays or control motors over Wi-Fi or USB (or both);
- USB to serial port adapter;
- rapid prototyping system with Wi-Fi, expansion port and three power channels;
- FlowBoard compatible with FlowStone 3 [5];
- something I have not imagined yet.

To make these different applications possible the board has been equipped with a switched mode regulator so it can be powered from a 7 V to 40 V DC power supply without wasting too much calories. The board can also be powered from

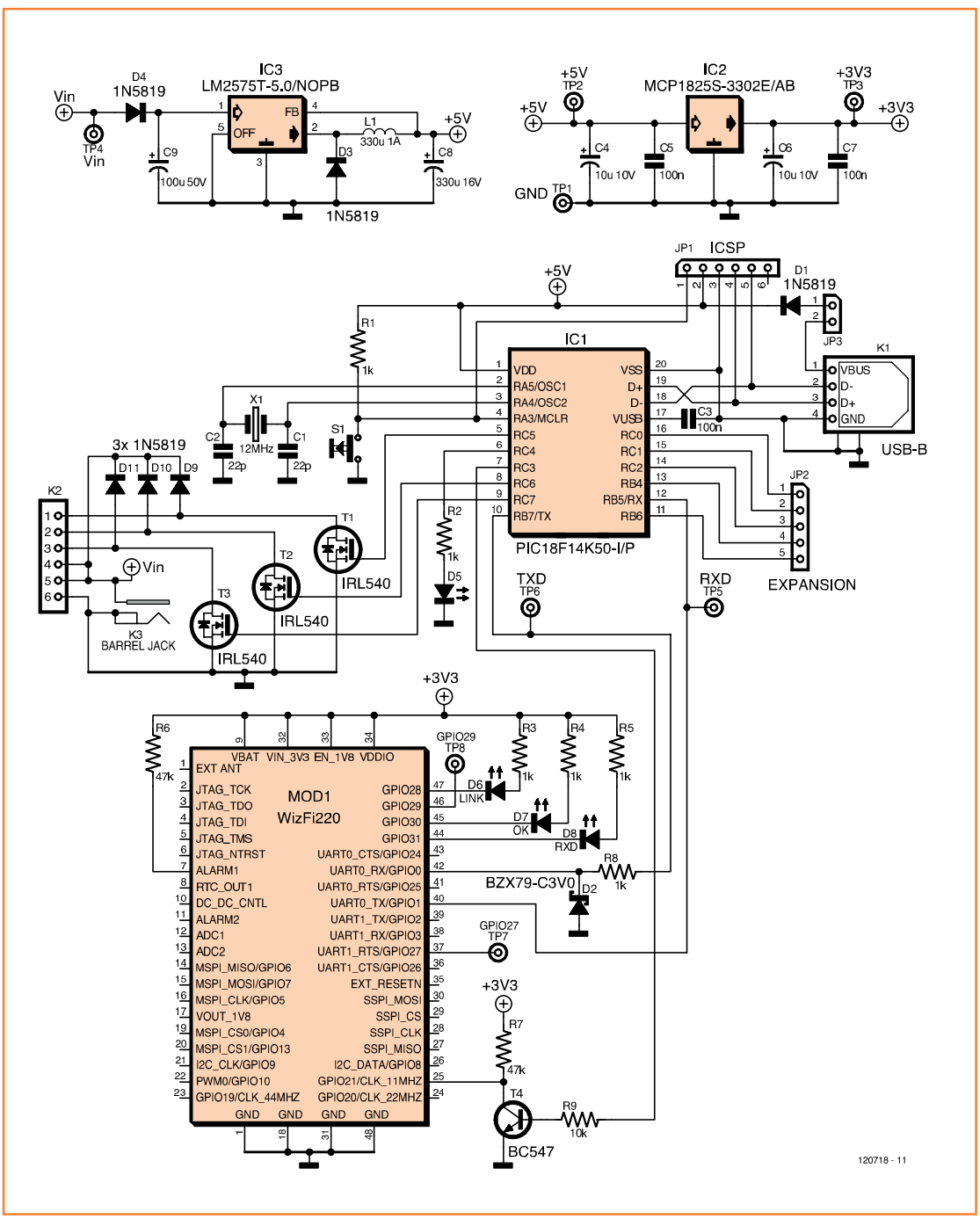


Figure 2.
Circuit diagram of the Wi-Fi Controller Board. Did you ever notice how people always connect the barrel jack the wrong way around? The center pin is supposed to be the common connection. I was wicked here on purpose because of PCB layout considerations.

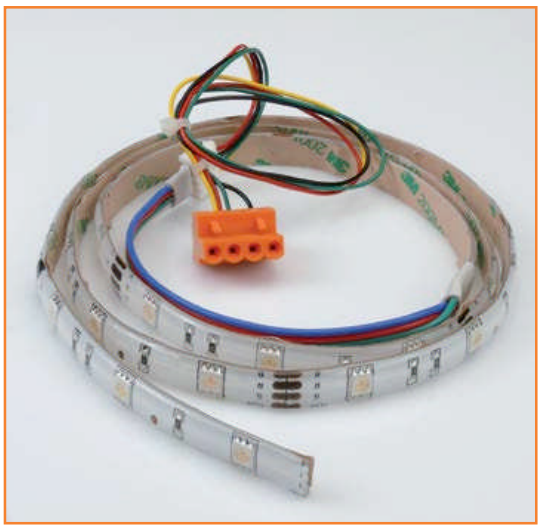


Figure 3.
A typical RGB LED strip with an adapter cable salvaged from a PC power supply.

the USB port, but keep in mind that the Wi-Fi module may consume quite a lot of power when it is transmitting. A large 3.3-V linear regulator provides power for the Wi-Fi module (due to current requirements you cannot use the 3.3 V regulator integrated in the MCU). The module is accompanied by three status LEDs indicating if the module is connected to an access point or not, and if data is being exchanged. Because the MCU is powered from 5 V and the Wi-Fi module from 3.3 V, level converter circuitry was added for the inputs of the Wi-Fi module. The MCU can handle 3.3 V-level input signals directly.

The MCU is clocked from a 12-MHz crystal oscillator to get the USB timing right. The pushbutton connected to the reset pin (MCLR) will serve mostly to activate bootloader mode (the external reset input must be disabled for this to work, fuse MCLRE=0FF). If the bootloader is not needed, this pushbutton can function as a reset button or it can be used for some other function.

The three conservatively rated MOSFETs feature an $R_{DS(on)}$ of 0.077 Ω, can switch up to 100 volts and are protected by flyback diodes so they can handle inductive loads too. As an output connector I have opted for a standard 4-pin .2'' (5.08 mm) pitch PCB terminal block even though most of the LED strips I have seen are equipped with a smaller pitch connector. However, since I do not know how standard these connectors are or if they are all wired in the same way, I preferred a more flexible solution. You can easily make an adapter cable from, say a floppy disk drive power cable (**Figure 3**).

For the power connector you can either use a center-pin barrel jack or a standard 2-pin .2'' pitch PCB terminal block. A diode provides a basic protection against polarity inversion.
A 5-pin extension header allows the use of the free MCU ports for custom purposes. A row of test points extends this connector in one direction; the in-circuit serial programming (ICSP) connector extends it in the other direction. All together they give access to ten MCU pins and all the power supplies. An LED connected to RC4 and a pushbutton are available too. Because the application program can be changed easily thanks to the bootloader and the USB interface, these options make the board into an excellent rapid prototyping platform.
All parts (except the Wi-Fi module) are standard through-hole parts so it should not pose any problems to assemble the board. I suggest you mount the Wi-Fi module first as it is a bit fiddly to position properly because of its many connections. Note that the voltage regulators should be mounted lying on their belly (see **Figure 4**, or on the solder side of the board). The reason for this is that if you don't use the proposed enclosure and you mount them standing up, they can easily be screwed to a heat sink (not that you really need one). The MOSFETs should lie on their back to fit in the enclosure.
If the LEDs must stick through the enclosure you should drill the holes first to get the length of their leads right. LED D5 (connected to RC4) and the pushbutton each have been positioned exactly under a standoff so that you can drill holes for them without tedious measuring first, just drill through the standoffs. The enclosure has a battery trap allowing you to mount Wi-Fi status LEDs on the solder side so you can see them only when you open the trap.
The pushbutton needs a cap of the right height. The 16-mm model will be almost flush with the top side of the enclosure.

To fit the PCB in the enclosure you need to trim in the top cover the four standoffs that keep the enclosure together. An easy way to do this is with a large drill bit. Do not trim too much or the screws won't have any material left to bite into. You should also break away the two bits of plastic meant to keep the battery in place.
The Wi-Fi module has a small connector for a better antenna (**Figure 5**). If you use it, just leave it dangling.

Programming the board

The firmware you can download for this project [6] contains the bootloader, the RGB LED strip application and the fuse settings in a single HEX file. Burn the HEX file into the MCU using an appropriate programmer (PICKIT, ICD or other) and you're set. With the bootloader programmed in the MCU you're ready to upload your own applications. The procedure is as follows:

- do not power the board, make sure JP3 is not in place;
- connect the board to a free USB port of your PC;
- press pushbutton S1 and keep it pressed while you place JP3. Actually, instead of JP3, I used a normally-closed (NC) pushbutton (microswitches are perfect for this). In this case, while you hold down S1, briefly press the microswitch to enter bootloader mode;
- The PC (Windows, Linux or Mac) should now detect the board as an HID. If this is the first time on a Windows PC, you have to show it the .INF file that comes with the download;
- on the PC launch the HIDBootloader tool provided in the download (Windows, download the Linux or Mac version from the Microchip website) (**Figure 6**). It should discover the board instantly;
- browse to your executable (HEX file) and click the program button;
- to quit bootloader mode and to start the application, click the Reset button or toggle the power supply of the board without holding down S1.

Configuring the Wi-Fi module

In [2] I showed you how to configure the WizFi220 module over a serial port using AT commands. This technique is useful if, for instance, you want the MCU to reconfigure the module on the fly. It also allows configuration of options not accessible in another way. The board presented here has a special mode to allow this way of configuring the module. The Wi-Fi module offers an easier way if all you want to do is connect to an existing Wi-Fi network. In this case you have to put the module in so-called Limited Access Point (LAP) mode. The board can do this as well.

For both methods you have to put the board in USB-to-serial mode first. To do so, connect RC2 to +5 V (for example, with a wire on the extension connector) and then connect the board to a free USB port of your PC. LED D5 starts blinking and you can send AT commands to the new virtual COM port (the number or name of which you will have to get from your operating system). If the Wi-Fi module is connected to an AP (i.e. when status LEDs 'LINK' and 'OK' are both on) type "+++" in a serial port terminal program to force the module into command mode (the 'OK' LED should be switched off). Now refer to [2] to configure the module.

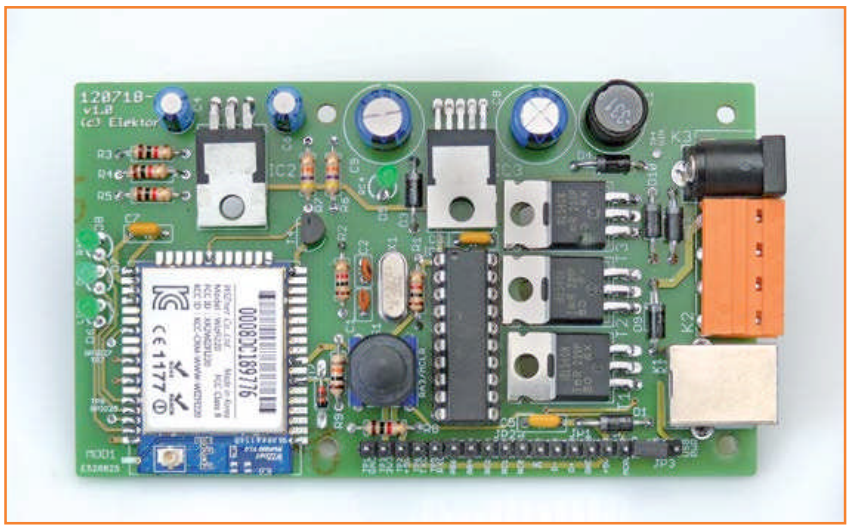


Figure 4.
The almost finished board, revision 1.0, fully assembled. Rev. 1.1 is identical except for some parts that have been moved slightly for a better fit with the enclosure.

To put the Wi-Fi module in LAP mode instead requires some dexterity. Watch the blinking LED closely, and get into its rhythm. When you feel ready, press pushbutton S1 when the LED is off and keep it pressed during two full blinks. Release the button only when the LED goes off after the second blink. If you managed to do this properly then after a second or so the three Wi-Fi status

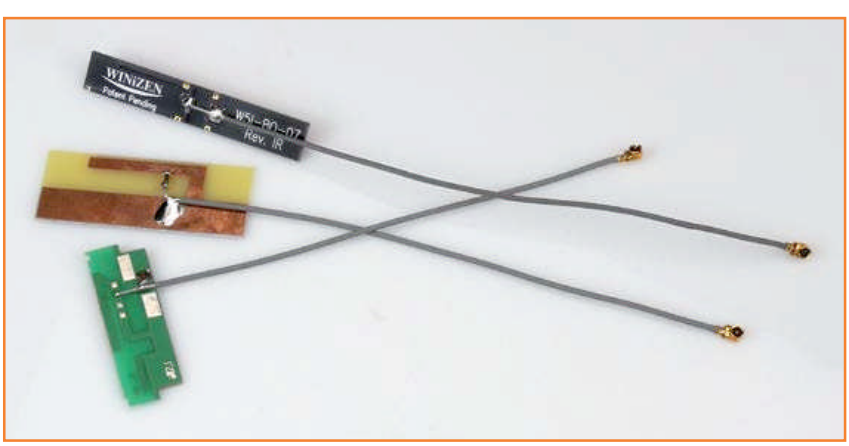


Figure 5.
Three 2.4 GHz antennas, one from Winizen (top) and the two others kindly provided by the girls from 2J (www.2j-antennae.com).

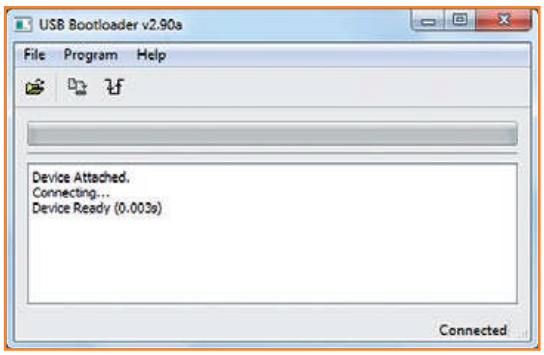


Figure 6.
Use this tool to upload a new application to the Wi-Fi Controller Board.

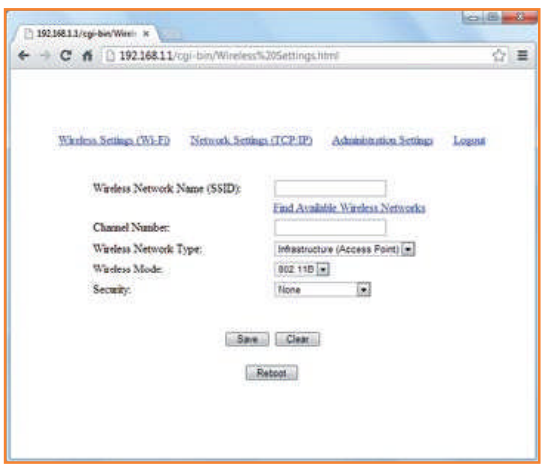


Figure 7.
The WizFi module's configuration page that's accessible in Limited Access Point mode makes life very easy.

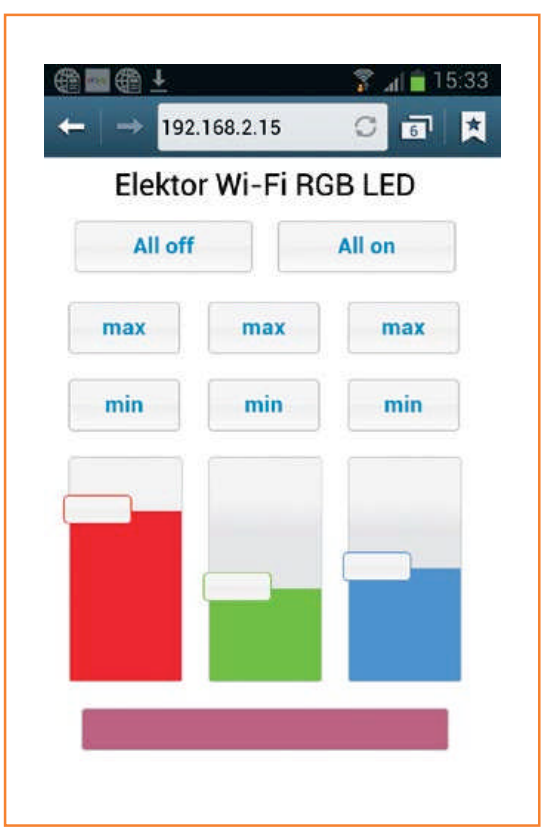


Figure 8.
The Wi-Fi Controller Board web page as seen from an Android smartphone. Is blue semi On or semi Off?

LEDs will suddenly flash rapidly several times and the module enters LAP mode. (If you count three blinks the module will restore its factory defaults, it's a quick way to get you out of trouble.) If you have the board connected to a serial port monitor you will see the message:

```
IP              SubNet          Gateway
192.168.1.1:    255.255.255.0:  192.168.1.1
[OK]
```

Now make sure that the module is in range of the AP you want to use with the board. Check for new access points with a PC, a smartphone or a tablet. If all is well you should see one labeled "WizFiAPxxxx" where xxxx is a number. Connect to it — it is an open AP so you don't need any pass phrases — and point a web browser to the address 192.168.1.1. You should see a page similar to **Figure 7**. Click on the link "*Find Available Wireless Networks*". A list will appear with the networks in range from which you can pick the one you want. Click "*Save And Continue*". This will take you back to the first form, which now shows the details of the selected network. Depending on the security settings of this network you can enter a pass phrase. Click the "*Save*" button to store the configuration.

Click the link "*Network Settings (TCP/IP)*" and fill in the form. I prefer to use a static IP for the module so I always know its address, but this is up to you. In the field "*S2W Connection method*" type "1,1,,80". This will turn it into a serial gateway listening on port 80, the default HTTP port used by web browsers. Again, feel free to use another value. When done click the "*Save*" button.

The "*Administration Settings*" link allows you to enter a password for the module. I have not used this option.

When done configuring click the "*Logout*" link. Now you will see the message "Rebooting..." and the Wi-Fi connection will be lost. The WizFi module will restart and try to connect directly to the selected network. If all goes well the LEDs "LINK" and "OK" will light up, meaning that you can now connect to the board. Enter the IP address of the module in the browser and wait until you get the page from **Figure 8**. When the page loads correctly you are ready to play.

It was fun to be able to control the lights in my house from the outside with my mobile phone. However, this is not how I plan to use the board. My intention is to hook it up to the motorized garage door so a smartphone becomes the key. It will then also be possible to give access to our home to someone even when we are (far) away. Indeed, this board has many possibilities, use your imagination!

(120718)

Internet Links

[1] Elektor Home Control:
www.elektor-labs.com/node/2325

[2] Wi-Fi/Bluetooth shield:
www.elektor.com/120306

[3] JQuery(UI):
http://jquery.com & http://jqueryui.com

[4] Arduino version:
www.elektor-labs.com/node/2373

[5] FlowStone (in this issue):
www.elektor.com/130064

[6] Firmware, Eagle PCB files, BOM; ordering:
www.elektor.com/120718

COMPONENT LIST

Resistors (5%, 0.25W)

R1,R2,R3,R4,R5,R8 = 1kΩ
R6,R7 = 47kΩ
R9 = 10kΩ

Capacitors

C1,C2 = 22pF, ceramic, 50V, 2.5mm pitch
C3,C5,C7 = 100nF, Z5U, 50V, 5mm pitch
C4,C6 = 10µF 63V, radial, 2.5mm pitch
C8 = 330µF 16V, radial, 3.5mm pitch
C9 = 100µF 50V, radial, 3.5mm pitch

Inductors

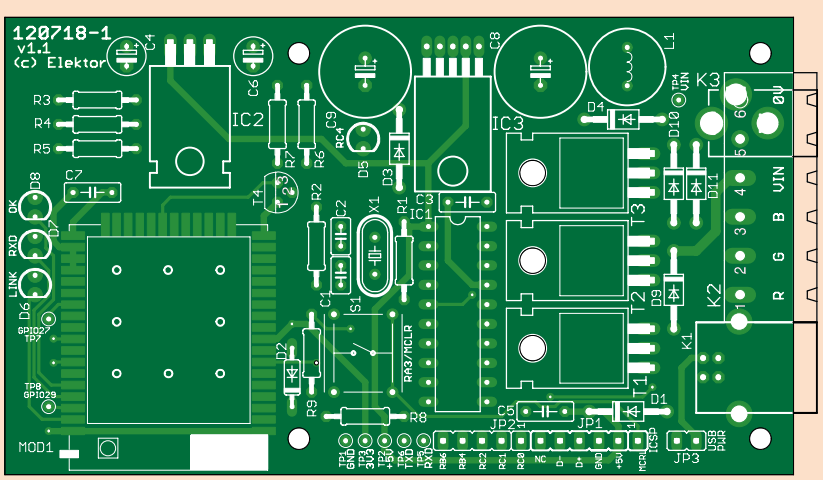
L1 = 330µH 1A, 5mm pitch, e.g. Würth Elektronik type 7447452331

Semiconductors

D1,D3,D4,D9,D10,D11 = 1N5819
D2 = 3V zener diode, e.g. BZX79-C3V0
D5,D6,D7,D8 = LED, green, 3mm
IC1 = PIC18F14K50-I/P
IC2 = MCP1825S-3302E/AB
IC3 = LM2575T-5.0/NOPB
T1,T2,T3 = IRL540
T4 = BC547

Miscellaneous

MOD1 = WizFi220 w. antenna, Elektor # 130076-92
JP1 = 6-pin pinheader, 0.1" pitch, vertical
JP2 = 5-pin pinheader, 0.1" pitch, vertical
JP3 = 2-pin pinheader, 0.1" pitch vertical
Jumper for JP3
K1 = USB-B receptacle
K2 = Connector 1x4, 90deg, 0.2" pitch, e.g. MSTBA4
K3 = DC socket
20-way DIP socket for IC1
S1 = Multimec type RA3FTL6 w. cap type S09-16.0
X1 = 12MHz quartz crystal, HC49/S case
Enclosure, Hammond type 1593QGY
PCB, Elektor # 120718-1



Recycle your ATX Power Supply

Elegant adapter board makes it easy

By **Ben Jordan** (USA)

This ATX bench top power supply adapter board allows you to convert any standard ATX computer power supply into a convenient supply for breadboarding and general electronics workbench use — simply and elegantly.

Features

- Outputs for ±12 V, 5 V, 5 V Standby and 3.3 V
- No ATX power supply hacking necessary
- Easy on and off switching of the connected ATX power supply
- LED status indication
- Capable of handling heavy supply currents
- Binding posts for each voltage output
- Slots for easy connection of alligator clips

For any electronics or embedded system tinkering, you have to have a good power supply. Now I don't know about your personal lab equipment budget, but mine is rather, well... let's just say that I'm married to the head of the finance department and she's not exactly an electronics engineer. You've probably crafted a few linear regulated power supplies over the years, but it's no trivial task to design and build a variable high powered lab supply. And also, if you're anything like me, the majority of what you tinker around with is digital circuits and low-voltage analog stuff, like audio preamps and such.

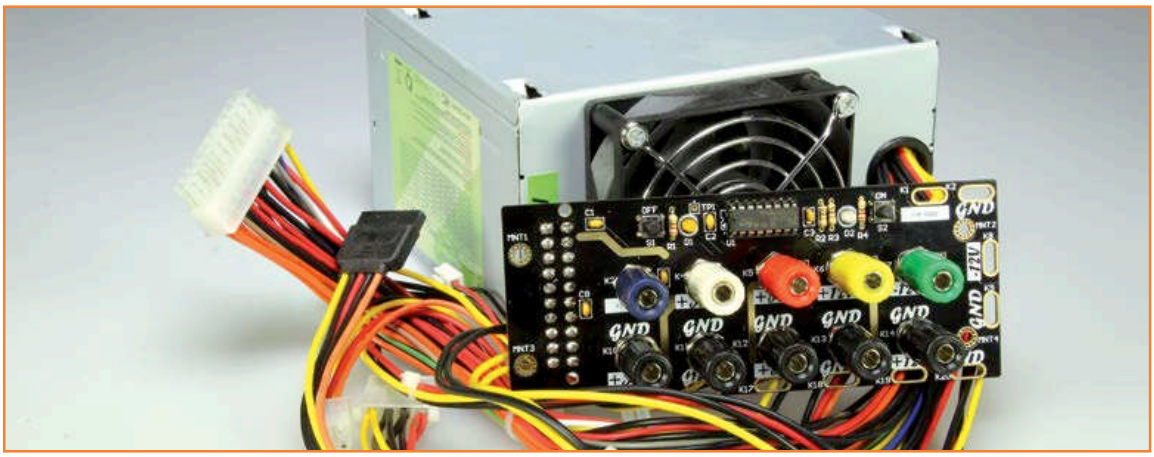
In most cases, I am messing around with a microcontroller and a few opamps, so typically it's handy for me to have a 5 V rail for the controller (and any glue logic I might have) and +/-12 V rails for the opamps. Increasingly, the sample devices I'm working with (for example the Freescale DSP56367 DSP or the NXP LPC2101 ARM-7 microcontroller) require low voltage rails for internal core and IO, namely 1.8 V and 3.3 V. This is a pain in the butt if you have to build a supply with all these rails for each project, but most of them (except 1.8 V) can be obtained from off-the-shelf PC ATX switching power supplies. This is great, because somehow (I don't exactly know how) over the years I have accumulated several of them. They all have 3.3 V, 5 V, 5 V Standby and ±12 V supply outputs. Granted, on many, cross regulation of the 12 V rails is not fantastic. But for the vast majority of opamp circuits it is more than adequate.

Design considerations

So, why not just put a bunch of banana jack binding posts directly on the power supply, as I've seen so many people do across the web? Good question! Here's why:

- ATX PSUs are designed with extremely tight clearances inside them. If you go putting binding posts in there, you may be closing in on safety clearances (and these are important. The AC power input side of most switching supplies sees voltage spikes over 1000 volts and DC busses around +370 volts. It's not worth risking life and limb!)
- You have a perfectly good computer power supply, and if you're anything like me you may want to use it to power op amps one day, and a computer motherboard the next. Why modify it beyond its original use?

I wanted a more elegantly designed solution, that not only had the binding posts, but also some clip points I could clip alligator leads onto.
So I set about designing a PCB for adapting the ATX supply to bench top use. As with anything, a good place to start is to define some specifications.



The adapter shall:

- not alter the power supply itself;
- use an ATX connector;
- have binding posts for each ATX power supply voltage and standby;
- have a matching return (GND) binding post for each voltage output;
- be capable of handling heavy supply currents;
- have a switch circuit to make use of the ATX on/off controller;
- have LEDs to monitor standby and Power Good signal, and
- use through-hole PCB design, so you, dear reader, can make one for yourself easily.

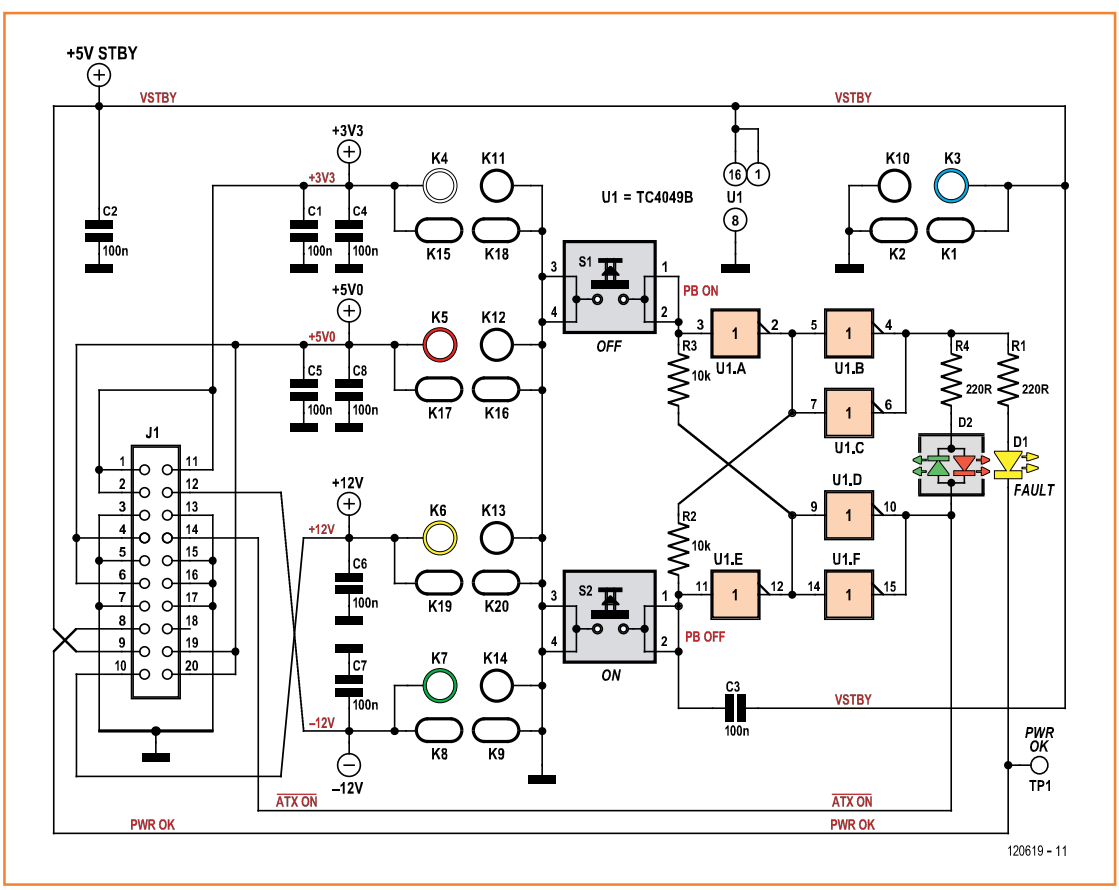


Figure 1.
The schematic shows how simple this power supply add-on actually is.

I had also considered a built-in panel meter so you could monitor the output voltage of each output, but I quickly realized this would be unnecessary feature-creep for this application, since the outputs are fixed and fairly well regulated, and most users (like me) can plug in their multi-meter and get a more accurate measurement anyway.

No complex circuit

Figure 1 shows the circuitry for this design. The switching circuit is a simple latch based on two

COMPONENT LIST

Resistors
R1,R4 = 220Ω
R2,R3 = 10kΩ

Capacitors
C1-C8 = 100nF

Semiconductors
D1 = LED, 3mm, yellow
D2 = LED, 3mm, 2-pin bicolor
IC1 = TC4049B

Miscellaneous
K3-K7,K10-K14 = binding post (socket) for banana plug
J1 = ATX connector for PCB mounting
S1,S2 = tactile push button

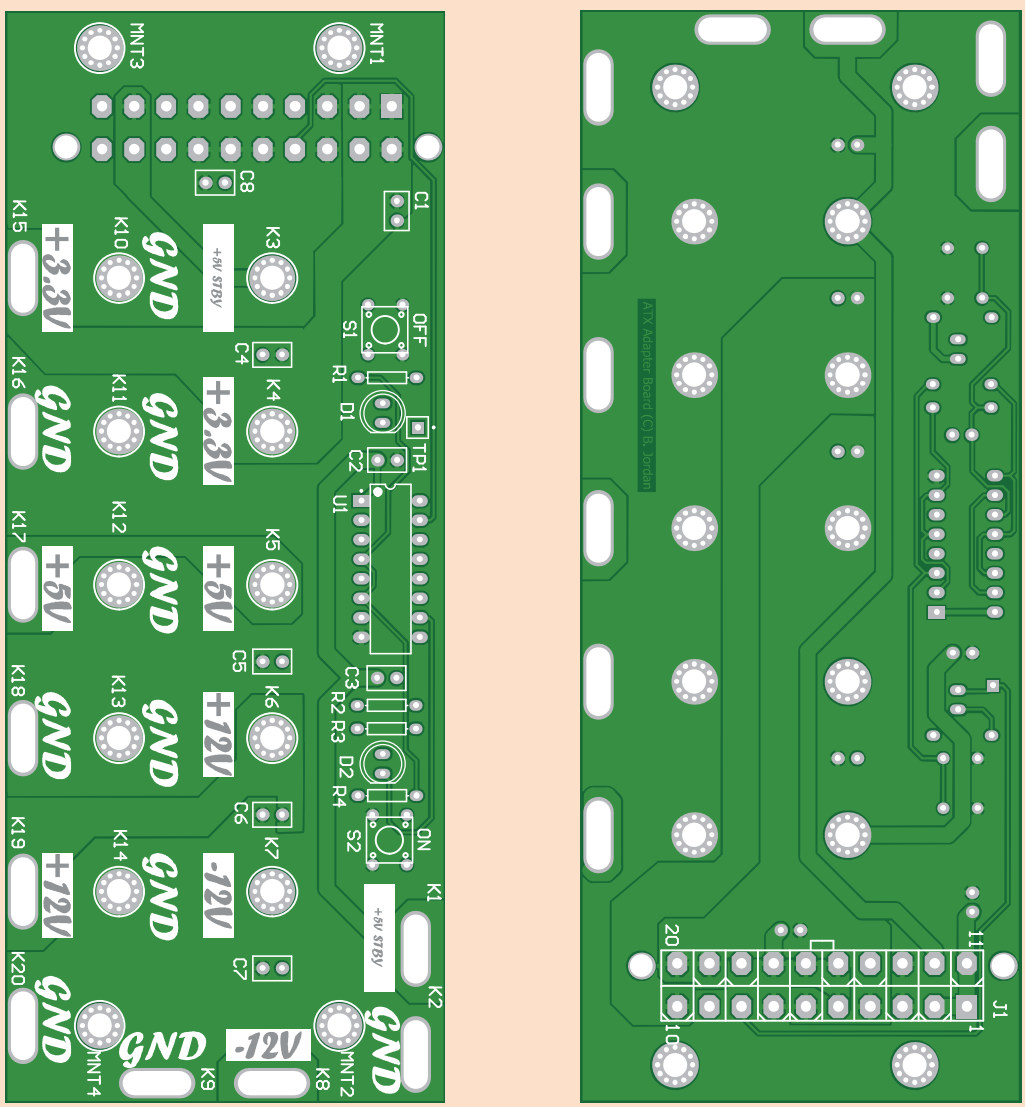


Figure 2.
The component layout leaves no room for misinterpretation. Note that the ATX connector should be mounted on the bottom side of the PCB.

inverters from a 4049 HEX CMOS inverter, powered from the ATX standby 5 V rail. The other gates in the inverter are used to drive the power on control signal to the PSU and the LEDs. Pressing S2 turns on the power supply while S1 turns it off again. D2, a bi-color LED, indicates standby mode (green) or powered-on mode (red).
ATX power supplies have built-in protection, but there's also a Power Good output that tells the motherboard when the PSU is ready, or if there's a fault condition. It would be nice to have this displayed on an LED. I used a yellow LED for D1 which indicates when Power Good goes high (Power Good is an active low signal). This way, you get some warning of ATX power supply fault conditions. In general, any continuous flashing is bad and your power supply probably has a problem.

All the capacitors provide a bit of decoupling for the different supply outputs, except C3 which is for de-bouncing the on-off circuit. All the capacitors are ceramic 100 nF types. R1 and R4 are LED dropping resistors and R2/R3 form the feedback paths for the on/off dual-inverter latch.

I have seen others converting ATX power supplies to bench top lab supplies and putting minimum load resistors in them. In my experience this is not necessary since the cooling fan is generally enough of a minimum load and most modern switching power supplies are designed to run down to zero load anyway. But if it makes you feel better, you can attach a 10 or 20 W resistor to the 5 V rail (10 Ω will usually suffice) by soldering the power resistor between the 5 V and GND binding posts at the rear.

Board design

This adapter board acts as a break-out board with an ATX header and the necessary on/off circuitry as well as standard-spaced binding posts for plugging in single or dual banana plugs. The silkscreen overlay as a guide for placement is shown in **Figure 2.** Note that the ATX power supply socket must be mounted on the rear side of the PCB, with all other components on the front side.
An additional novelty of this board is the unique plated slots around its edge, which are used to clip on 'alligator' style test leads as well. The silkscreen text indicates the output voltage of each one. The board layout is available as a PDF download from [2].

The plated-through holes designated K3 to K14 are for bolting the back conductive end of the banana socket binding post onto. Notice the use of multiple vias in a circle around each of these and also around the mounting holes. These offer increased strength for tightening up screws over them and also for the binding posts a good low-resistance current path through the PCB.

In practice

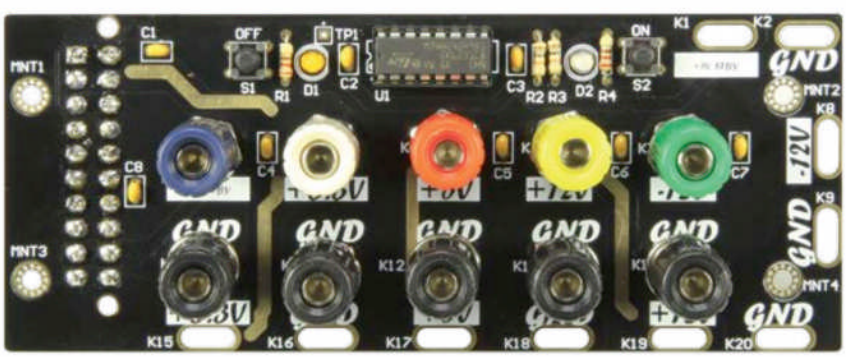
This ATX adapter board offers standard 0.75" jack spacing for dual-banana plugs and has color-coded binding posts for each of the voltage rails: 5 V standby, 3.3 V, 5 V, +12 V and –12 V. I have now been using this board extensively at work and at home, powering several development kits off a single ATX supply. At work I have an Altium Nanoboard-II and two Altium Nanoboard 3000 FPGA development boards all running at once from one power supply, as well as a few other things. It has really simplified things well for me!

(120619)

Internet Links

[1] http://jordandsp.com/ATX-bench-top-power-supply-adapter.php

[2] www.elektor.com/120619

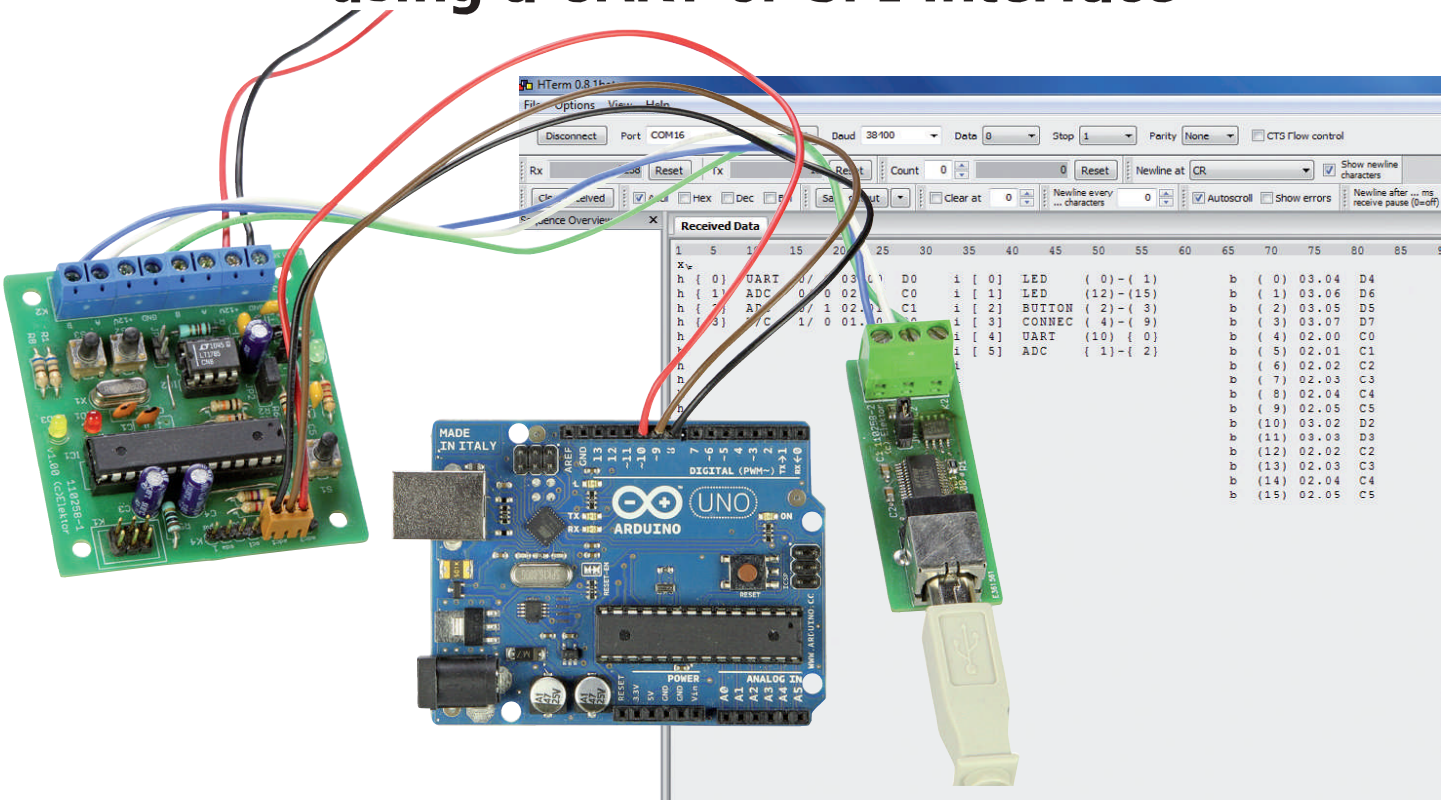


I want it

This project is available in limited supply as a complete kit, with the PCB, all components including binding posts, components and an ATX connector, plus comprehensive assembly instructions which include the schematic diagram and PCB assembly diagram, as well as drill templates for panel mounting. Visit [1] for more information on price and ordering.

Off to the EF Library!

Controlling hardware using a UART or SPI interface



By **Jens Nickel**
(Elektor Germany Editor)

If a suitable protocol is defined, hardware connected to your PC can be controlled using a terminal emulator program. The necessary firmware can be rustled up quickly using Elektor's 'Embedded Firmware Library' (EFL), and the code can be written without knowledge of whether a UART or some other interface will be used for the connection. The protocol described here is ideal for experimentation and development purposes.

In the previous edition we described our modular 'Embedded Firmware Library' (EFL), written in the C programming language. This helps beginners and old hands alike to develop code for an embedded project that is independent of the underlying hardware and which can therefore easily be ported from board to board and from microcontroller to microcontroller. This is achieved using a hardware abstraction layer consisting of a code file for the board and a code file for the microcontroller.

However, the idea of modularity is taken even further in the EFL. Protocol libraries allow programs that communicate to be written without knowledge of the communication channel that will be used: it makes no difference whether commands and data are transferred using a UART and

RS-232 or RS-485 connection, or using TCP/IP over Ethernet. Just a couple of lines of the application program code need to be changed to use a different interface. In this article we will look at this in more detail and present a protocol that can be used in simple hardware control applications and for development.

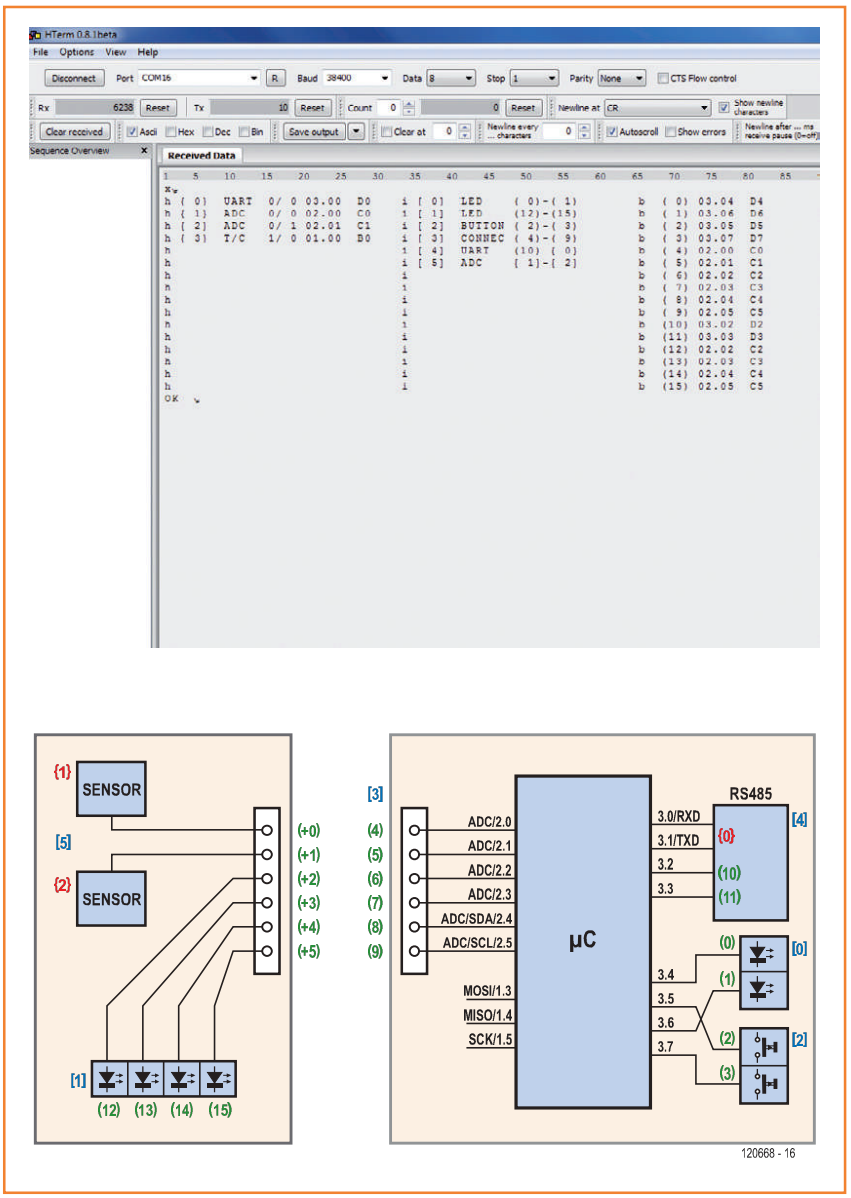
Mini protocol

When trying to get code up and running simple tests such as turning a LED on and off or reading a digital input can be very helpful. Often, however, a microcontroller board will not have any buttons to help you carry out such tests. One alternative option is to control the board from a PC, for example over a serial interface. If the protocol is restricted to ordinary printable ASCII characters it is easy to use a terminal emulator program to send commands to the board, avoiding the need for special-purpose software on the PC.

Our mini protocol (called 'BlockProtocol') is designed to let us set and clear pins on the microcontroller from the PC terminal. If there is an LED connected to the pin in question, this gives immediate feedback; otherwise a multimeter or oscilloscope can be used. There are also simple commands for obtaining the status of digital inputs and for reading ADC conversion results from the analogue inputs (see the text box 'BlockProtocol'). The code in the accompanying library module can of course be extended in many ways as required. Perhaps a reader will be inspired by the control protocols designed by Andreas Eppinger [2] and Uwe Altenburg [3], which offer a much wider range of facilities.

Map, blocks and board pins

The BlockProtocol includes the 'x' command specially designed for use in development with the EFL. The command causes the board to send a dump of the EFL tables to the PC. The tables are used by the hardware layer to determine the microcontroller pins and registers that need to be changed in response to a function call. The entries are specific to the board and any expansion board that might be attached. The screenshot in **Figure 1** shows what appears in the terminal window when the board shown alongside is used. On the left is the 'map' including the features of the microcontroller such as the UART and ADC that are set up by the board initialization code.



The peripheral blocks are shown in the middle, while on the right is a list of the pins on the board. This is particularly important information for anyone planning to adapt or write an EFL board file for a new board. More extensive documentation on the internals of the EFL are given in the extra document that can be downloaded at [1].

Control via UART

We shall now show by example how easy it is to use the protocol in your own microcontroller applications. We will use an experimental node, familiar from our series on the ElektorBus. This is a very compact board based on an ATmega328. As usual we will connect the node to the PC using its

Figure 1.
Dump of the EFL tables as seen in the terminal emulator. On the left is the map with the microcontroller features in use (red numbers in the board block diagram); in the middle is the block table (blue numbers); and on the right are the board pins (green numbers).

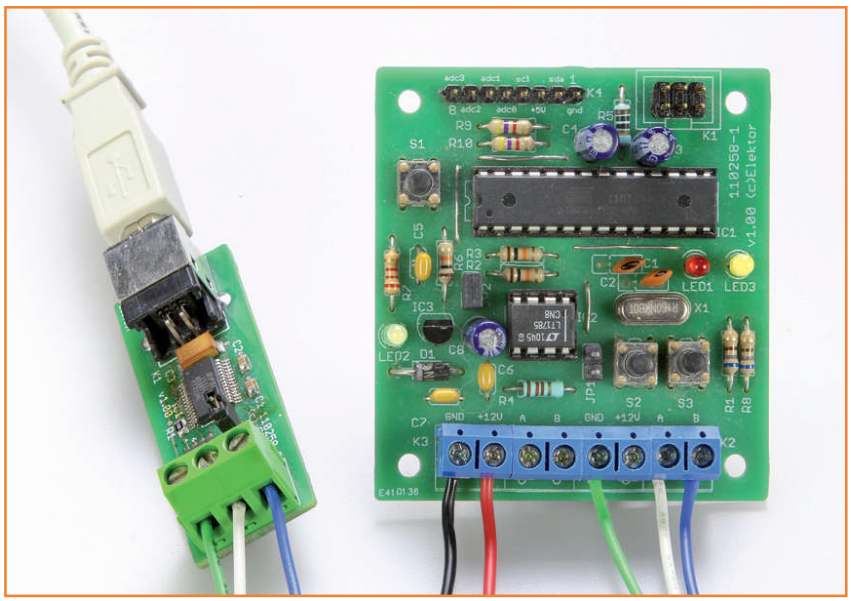


Figure 2.
Our small microcontroller board can be controlled over a UART/RS-485 interface with the help of a simple ASCII text-based protocol and a terminal emulator program.

RS-485 interface and a USB-to-RS-485 converter (**Figure 2**). Instead of the ElektorBus protocol we will use our new text-based BlockProtocol.

The necessary firmware for the microcontroller can be downloaded at [4]; the application can also be found in the EFL code base at [5]. An overview of the files is given in the extra documentation mentioned above. Double-clicking 'ExperimentalUART.atsln' opens the project in Atmel Studio: see the screenshot in **Figure 3**. On the right-hand side is a list of the files in the project. The files Controller.h, Controller.c, Board.h and Board.c form the hardware layer.

Listing:
Basic structure of an EFL application.

```
int main(void)
{
  Controller_Init();
  Board_Init();
  Extension_Init();

  ApplicationSetup();

  while(1)
  {
    ApplicationLoop();
  }
};
```

In this case the code is the same as in the EFL example software from the previous issue, as we are using the same board and the same microcontroller. The 'Libraries' directory includes the files UARTInterface.h and UARTInterface.c, which form the library for the UART interface on the board. Here this is our RS-485 driver (corresponding to the 'physical layer' of our communications). There is also a module implementing our protocol (BlockProtocolEFL.c and BlockProtocolEFL.h). Together these two library modules provide a defined software interface which we will now look at.

A little code

In the main source file of the application code itself we need to include the library header files as follows.

```
#include "UARTInterfaceEFL.h"
#include "BlockProtocolEFL.h"
```

The main function in an EFL-based project is always structured in the same way: see **Listing 1**. In the application set-up function, which is called whenever the application starts up, we initialize the libraries:

```
UARTInterface_LibrarySetup();
UARTInterface_SetBaudrate(0, 38400);
BlockProtocol_LibrarySetup(UARTInterface_
Send, 0, UARTInterface_GetRingbuffer(0));
```

Here the second line sets the baud rate of UART interface block 0 (which is where we have our RS-485 driver connected) to 38400 baud. The third line needs a bit more explanation. We are telling the BlockProtocol library that it should use the function UARTInterface_Send when it wants to send data from the board. The second parameter is the number of the UART interface block to be used (even though on the experimental node there is only one UART interface).

The third parameter is a pointer to a 'ring buffer' (or 'circular buffer') to be used to store received bytes. Here the required pointer is obtained as the return value of the function UARTInterface_GetRingbuffer(0), which is also implemented in the UARTInterface library.

In the main application loop function, which is called regularly when the application is running,

we need just a single line:

```
BlockProtocol_Engine();
```

This is the main function in the BlockProtocol library. It checks whether characters from the PC have been received in the ring buffer, and as soon as a <CR> (ASCII 13) is seen it is interpreted as the end of a command and the command is then carried out. A response is assembled, which is either simply 'Ok' or a value such as 'HIGH' or 'LOW', or information about the EFL variables in table form.

Let's try it

Having compiled the code and flashed it into the board we can try it out using a terminal emulator program such as HTerm [6]. First we have to specify which COM port to use and the baud rate. In the 'input control' window we have to set the program up so that the <CR> character is sent when the Enter key is pressed (see **Figure 4**).

We will first use the library at its lowest level to demonstrate direct access to to the output port pins of the microcontroller. From the circuit diagram of the experimental node [7] we can determine that the red LED on the board is connected to port pin PD4. Port D on the AVR microcontroller corresponds to port index 3. And so we simply type into the terminal emulator

```
p 3 4 + <ENTER>
```

and the red LED on the board will light.

As another example, using commands like 'p 2 0 +' or 'p 2 0 -' we can set and clear pin PC0 on the expansion connector. If an expansion board, such as the sensor/LED board form the previous issue or the relay board from the ElektorBus series [8], is fitted this gives us a simple way to control external hardware devices from the PC. Instead of the terminal emulator program we could run a dedicated program on the PC to send commands like 'p 2 0 +' to the appropriate COM port in order to switch the relay.

Alternatively we could connect a smartphone to the experimental node using the Andropod RS-485-UART bridge board [9] and write a small Android app. The software at [10] could be used as a starting point for such a project, replacing the bytes of the ElektorBus protocol with the character sequences given above.

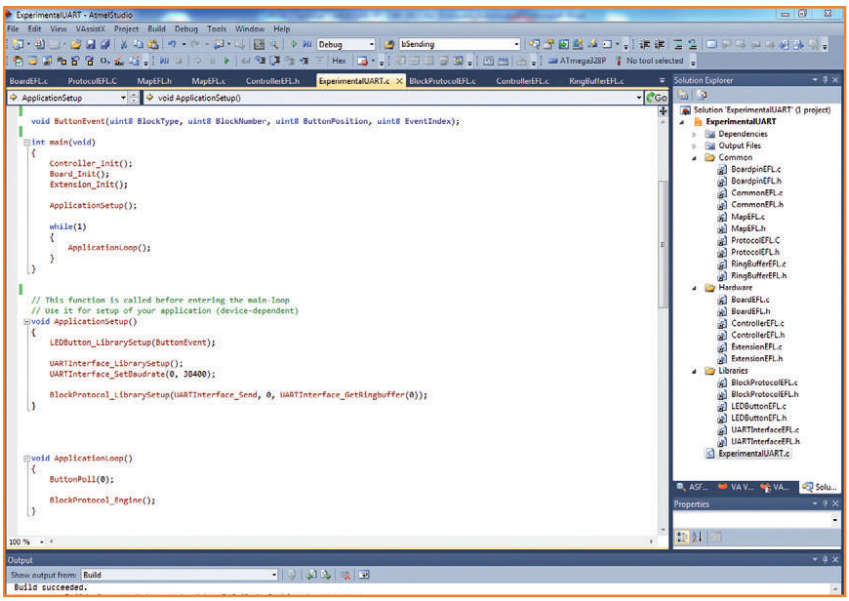


Figure 3.
Code to control hardware over a UART interface.

Higher-level commands

Directly controlling port pins is not the main raison d'être of the EFL. If we want to port our device control program to use a different microcontroller board with different wiring, we would have to change the character sequences given above inside the PC or Android software.
For this reason the mini-protocol also allows hardware-independent control of digital inputs and outputs, just like when writing code to run

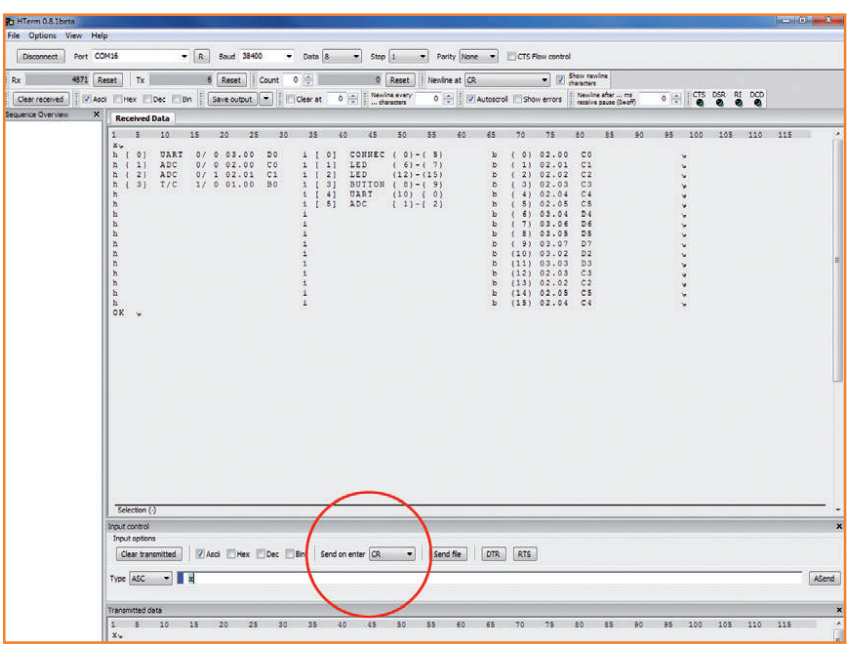


Figure 4.
The HTerm terminal emulator lets you choose what is sent when the Enter key is pressed.

on the microcontroller itself as described in the previous issue.
With the string

```
L 0 1 + <CR>
```

we can switch on LED 1 in LED block 0 (which on our board is the yellow LED). With

```
B 0 0 ? <CR>
```

we can request the status of the test button (button 0 of button block 0). And with

```
R 0 0 + <CR>
```

we can turn on a relay.

BlockProtocol

Each command starts with a single character, possibly followed by one or two decimal values. These specify to the EFL the pin on which the command will act. A final character is used to specify any particular action to be performed.

x
Outputs the EFL map, block and board pin tables (see additional documentation at [1]).

p *x y* +, p *x y* –, p *x y* ?, p *x y* #, p *x y* *
For microcontroller pin *y* on port *x*: set high; set low; read state; read ADC conversion result (if applicable); toggle under timer control.

b *x* +, b *x* –, b *x* ?, b *x* #, b *x* *
For board pin with index *x* in the board pin table: set high; set low; read state; read ADC; toggle.

i *x y* +, i *x y* -, i *x y* ?, i *x y* #, i *x y* *
For pin in position *y* within the block with index *x* in the block table: set high; set low; read state; read ADC; toggle.

C *x y* +, C *x y* –, C *x y* ?, C *x y* #, C *x y* *
For pin *y* of connector *x*: set high; set low; read state; read ADC; toggle.

L *x y* +, L *x y* –, L *x y* ?, L *x y* *
For LED in position *y* in LED block *x*: set high; set low; read state; toggle.

R *x y* +, R *x y* –, R *x y* ?
For relay in position *y* in relay block *x*: set high; set low; read state.

B *x y* ?
For button in position *y* in button block *x*: read state.

A *x y* #
For ADC in position *y* in ADC block *x*: read conversion result.

*
Stop timer-controlled toggling.

Three-wire interface

The approach described above assumes that we have external access to a UART on the microcontroller, for example via RX and TX pins brought out on the board. We also need a more-or-less complete EFL microcontroller description file which describes not only the I/O pins and ADC functions but also the UART functions that are available.

When starting out with a new microcontroller (for example because you want to develop an EFL microcontroller description file for other readers to use) it is probably simplest to start with the I/O functions. The first major hurdles are to study the datasheet to determine how to go about setting the level on the I/O pins and reading their status, and to find at least three general-purpose I/O pins that are brought out to convenient points: fortunately most boards satisfy this condition. It will then be possible to control the board using the protocol described above, over a three-wire SPI interface implemented in software. One of the wires is a clock signal, the second carries bytes from the master (the PC) to the slave (the board), and the third carries bytes in the opposite direction.

The SPI specification allows for the further possibility of having communications over the three-wire interface initiated by the slave rather than by the master. In this case each participant takes its 'out' line high and waits for the other participant to do the same. Then communication can begin, with, as usual in SPI systems, the master providing the clock and data bytes flowing in both directions simultaneously. When there are no more data to send a 'stop character' (<LF>, ASCII 10) is sent. When both participants have sent this byte, the communication is complete.

We have encapsulated this protocol in the EFL module called 'ThreeWireInterfaceEFL'. With the help of this module it is easy to set up communi-

cations using the software SPI interface in place of the UART interface.
The code required in the application set-up function is now as follows:

```
ThreeWireInterface_LibrarySetup();

BlockProtocol_
LibrarySetup(ThreeWireInterface_Send, 0,
ThreeWireInterface_GetRingbuffer(0));
```

These lines tell the BlockProtocol library that we will be using the three-wire interface to send and receive data: compare them with the code for setting up the UART interface above.
In the main application loop we need the command

```
ThreeWireInterface_Listen(0);
```

which checks whether the other device has taken its data line high to indicate a request to send data. If so, the communication is set up. When using the UART for communication we do not need a listener function like this, as in that case a microcontroller interrupt can be used to detect when characters arrive and automatically store them in the ring buffer.

Making contact with an Arduino

To demonstrate the EFL concept we selected an Arduino Uno board. The code for this board can be found in the project ArduinoUnoEFL [4][5]. We decided to use general-purpose I/O pins PB0, PB1 and PB2 to access the board: these are brought out on the 'digital' I/O connector as Digital8 to Digital10. A simple three-way cable (for example using Conrad order code 741221) provides a practical solution (see **Figure 5**).
Of course we also need a connection to the PC at the other end, and that will also need to have a three-wire interface. We pressed an experimental node board into service as a gateway, converting from three-wire format to RS-485 (and thence to USB) and back again. The firmware involved can also be downloaded at [4] or [5]: the Atmel Studio project can be found in the directory 'ExperimentalSPI'.
The gateway function has its own library module called OneToOneGatewayEFL. With a call to

```
OneToOneGateway_Engine();
```

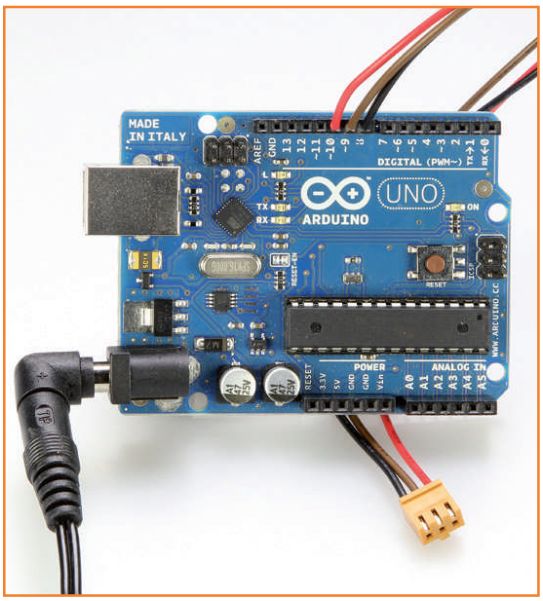


Figure 5. Communication with an Arduino Uno over a three-wire interface.

inserted in the main application loop the processor will regularly check whether a string, terminated with a <CR> character, has appeared in one of the input ring buffers. If so, then its contents are sent out on the other communication channel. Then the other input ring buffer is similarly checked, and so on.

It should almost go without saying that the gateway module itself is programmed in a manner independent of the underlying communication channels and can therefore easily be used to perform other similar gateway functions. The function OneToOneGateway_LibrarySetup takes two triples of parameters to specify the two communications channels being connected. In this example the call appears as follows:

```
OneToOneGateway_
LibrarySetup(UARTInterface_Send,
0, UARTInterface_GetRingbuffer(0),
ThreeWireInterface_Send, 0,
ThreeWireInterface_GetRingbuffer(0));
```

We now just need to plug the cable into the experimental node's expansion connector (pins PC0 to PC2, see **Figure 6**) and flash the appropriate hex file into its microcontroller.

The pins used to form the three-wire interface have to be specified separately for the two different boards. The three-wire interface on each device is a separate peripheral block: recall that

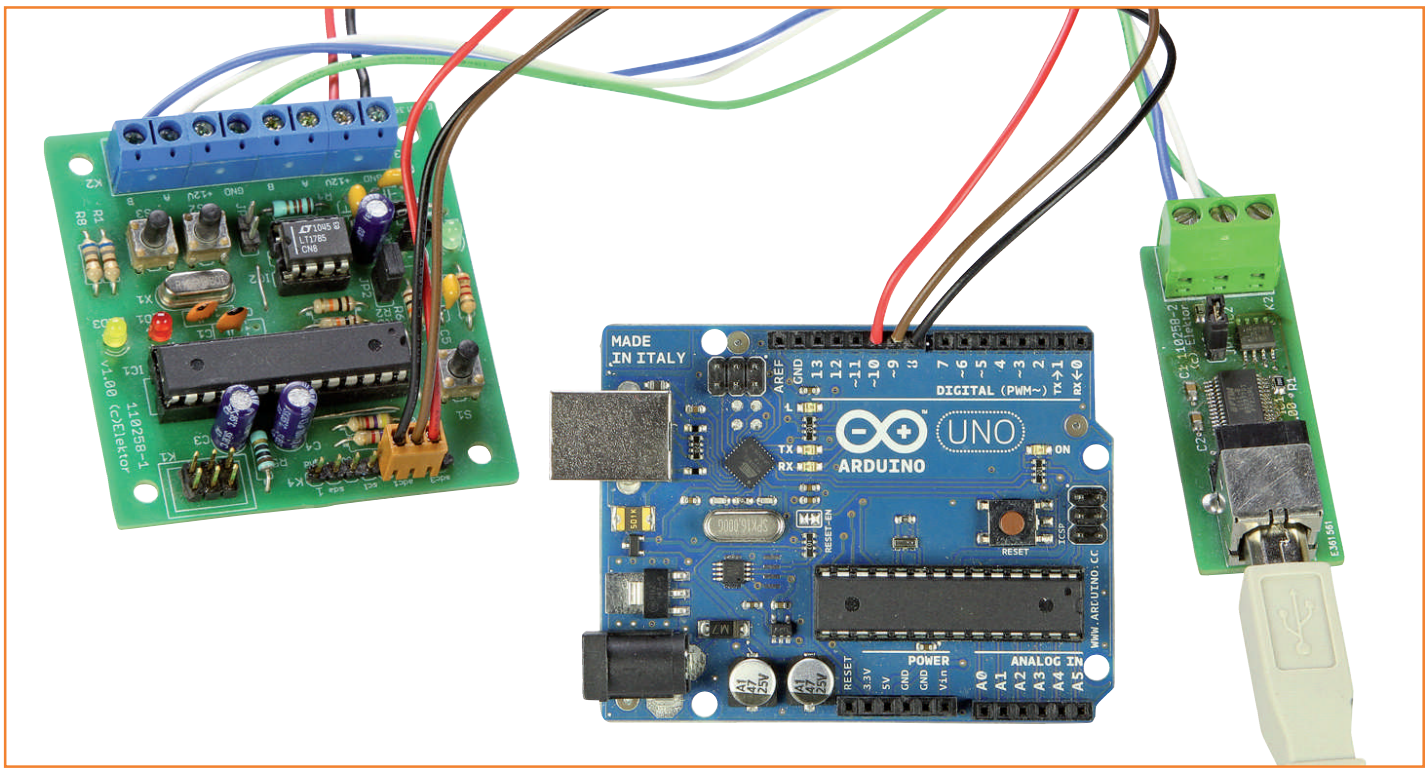


Figure 6.
The experimental node acts as a gateway between the three-wire interface and the UART/RS-485 interface.

the board file encapsulates the mapping between peripheral blocks and microcontroller pins. The configurations for the three-wire interface, including these pin assignments, can be found in the Board_Init function in the file BoardEFL.c.

It rocks!

In the terminal emulator program we first issue the 'x' command. The Arduino board responds directly, sending the ELF variables out over the three-wire interface, albeit at a rather more leisurely pace than before (the speed is comparable to that of a 9600 baud serial port).
The command

```
C 0 13 + <CR>
```

can be used to set pin 13 of the Arduino's 'digital' connector high: this pin drives an LED on the Arduino Uno board, and so we get immediate visual feedback.

In the next installment we will take the EFL project further: suggestions and contributions are more than welcome.

The most recent version of the EFL code can be found on the Elektor.Labs pages at [5].

(130154)

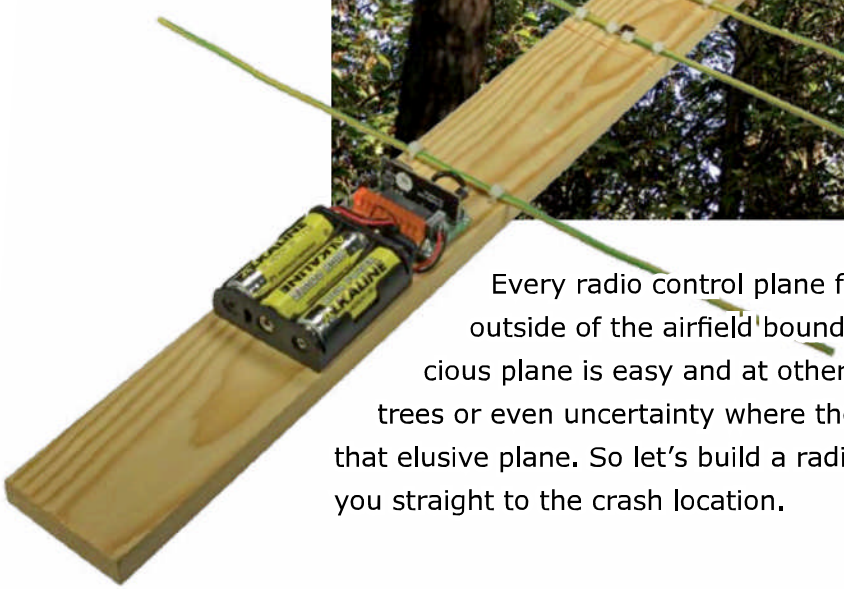
Internet Links

[1] www.elektor-magazine.com/120668
[2] www.elektor-magazine.com/100576
[3] www.elektor-magazine.com/120296
[4] www.elektor-magazine.com/130154
[5] www.elektor-labs.com/efl
[6] www.der-hammer.info/terminal/index.htm (in German only)
[7] www.elektor-magazine.com/110258
[8] www.elektor-magazine.com/110428
[9] www.elektor-magazine.com/110405
[10] www.elektor-magazine.com/120097

Lost Model Finder

Straight to the crash site!

By **Robert Budniak**
(Australia)



Every radio control plane flyer at some time has had a plane go down outside of the airfield boundaries. Sometimes the location of the precious plane is easy and at other times they are hard to find. High grass, trees or even uncertainty where the plane went down all make it hard to find that elusive plane. So let's build a radio direction finder (RDF) that ideally takes you straight to the crash location.

There are a number of systems around used to locate lost R/C models. I considered that I could build a system that was a little bit better. The concept specifications were:

- Lightweight transmitter for the plane
- Battery backup in case of disconnection of the main battery
- Using commercial, type approved UHF ISM band radio modules
- Range at least 600 ft. (200 m)
- Receiver to be handheld and able to use radio direction finding (RDF) to locate the model.

From this I was able to put to together the design described here. If you're strong in model building and flying, but not in electronics, consider doing this project as a club activity.

Features

- low cost design
- TX board in SMD, RX board in T/H technology
- Suits most 315/433 MHz ISM short range radio modules
- Programmable personal callsign on each TX
- Minimum range: 600 ft (200 m)
- Maximum range: depends on terrain and TX and RX modules used
- 4-element Yagi directional antenna on RX

Transmitter

Every country has a series of radio frequencies allocated to the Instrument, Scientific and Medi-

cal Band (ISM), sometimes combined with 'Short Range Devices' (SRD). These frequencies vary between countries. While in the USA 315 MHz is used, in Australia and most of Europe we find 433 MHz. Because there is a demand for simple and cheap transmitters, there are a number of manufacturers supplying RF modules to this market. The type of module used in this project uses Amplitude Shift Keying (ASK), and these types of modules seem to have a standardized pin configuration from different manufacturers. So that's the RF transmission part taken care of. Note that the parts list for the project indicates suggested types only.

As you can see in the schematic in **Figure 1**, the RF module just needs to have a data stream applied its 'DATA' pin, and for this project I used a simple PICAXE08M chip. I chose this microcontroller as it was readily available, the programming language was easy to learn, and the chip did not require any special equipment to program it. Really an ideal system for low volume and low complexity projects.

The code is only a few lines (**Listing 1**), and in the sample code for this project the coded generates three short 500-Hz tones and then a pause of about 2.5 seconds. Whilst the code can be copied, I would recommend that the line where the tones are generated is changed if you use a number of these transmitters near each other (like in a club environment).

The transmitter is normally powered off the plane's receiver. However sometimes the plane's battery gets disconnected in an accident, so a backup battery is provided. The backup battery is a small Li Po battery (around 130 mAh capacity) that usually powers small indoor planes. This battery is available cheaply from online suppliers.

Usually you'd use steering diodes to automatically select the power supply. However with the low voltages used in this project (5 volts and 3.7 volts) even the 0.4 volt drop of a Schottky diode was a sizable proportion of the voltage. Looking through some online databases, I found some small signal transistors type DTB123YK (T1, T2), which also had base-emitter and base resistors in the chip. They work quite well in steering the voltages of the batteries, and the voltage drop across the transistor is just 100 millivolts. The only down side is that when the main battery is disconnected the backup battery takes over. Hence the backup battery needs to be disconnected when the plane is not in use, or it will run down.

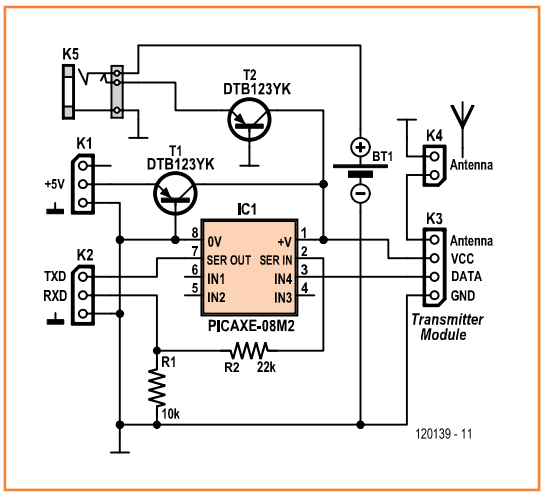


Figure 1.
Circuit diagram of the transmitter, which is not much more than a programmed PICAXE microcontroller.

Listing 1. PICAXE TX code

```
main:
sound 4,(0,10,120,10,0,10,120,10,0,10,120,10,0,10)
high 4
pause 2300
goto main
```

So that the Lost Model Finder can be left permanently *in situ* (it's cheap enough that one can have one for each plane), it's designed with a charging port combined with an on/off switch. This is achieved by using the switch in a 2.5-mm jack socket that's operated by a 2.5-mm jack. With the jack inserted, the backup battery is disconnected. If a jack that is connected to a charger is inserted, then you can charge the battery.

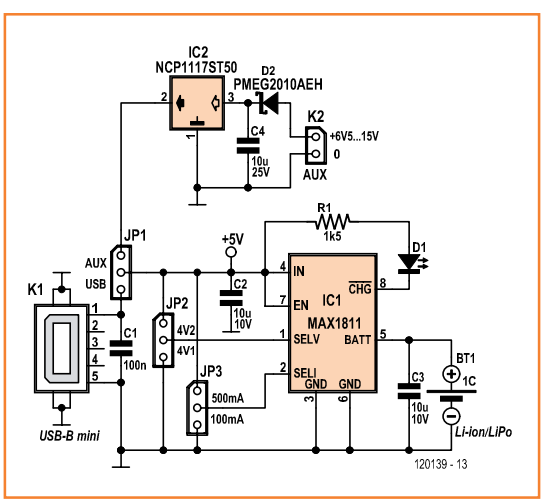


Figure 2.
A possible charger circuit for the LiPo/Li-Ion battery in the transmitter. It's either externally powered (6.5–15 VDC), or USB powered as selected on JP1.

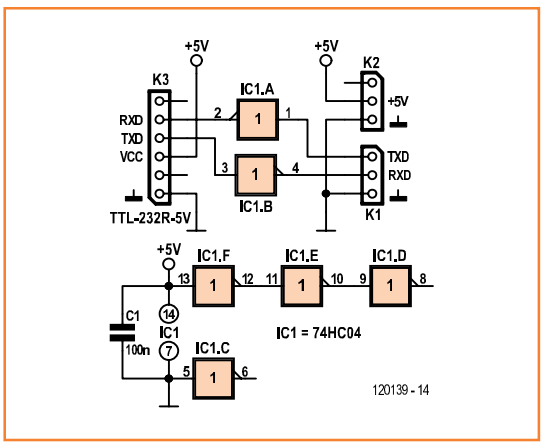
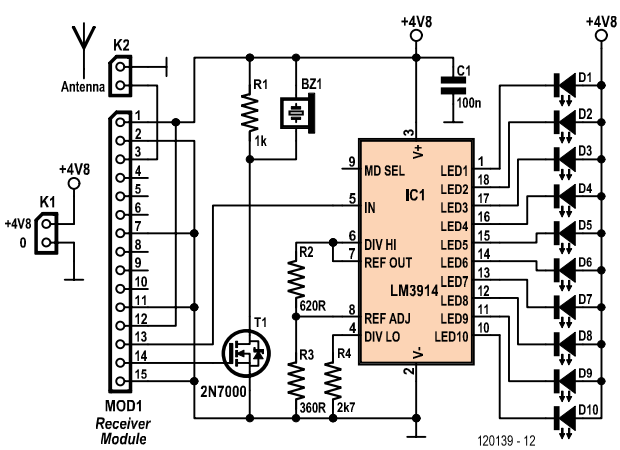


Figure 3.
Build this inverter circuit if you use an FTDI TTL-to-232 adapter between the transmitter's TX/RX pins and, say, a microcontroller.

Charger

A suggested circuit for a charger is shown in **Figure 2**. Two inputs control the mode the charger is operating in. One input selects the regulating voltage, 4.1 V or 4.2 V (jumper JP2, SELV). The other input sets the charge current, 100 mA or 500 mA (jumper JP3, SELI). A nice feature of this IC is the ability to precondition a near-dead battery before charging. The enable input (EN) is not used and thus permanently connected to the power supply. The general description in the datasheet states specifically it can be powered from a USB port and can handle input voltages as low as 4.35 V, the minimum of a USB port. With higher input voltages (the MAX1811 can handle 6.5 V max.) and with the high current set the IC will limit the charge current to keep the die temperature at a safe level. In case only a power source with a higher voltage is available a low-dropout 5V regulator is added (IC2). A jumper selects the input voltage for the MAX1811 (JP1: AUX or USB). Do not connect an input voltage to the regulator if the USB port is selected as input source. The input decoupling of the MAX1811 (C2) doubles as the regulator output decoupling. This was done to save space.

Figure 4.
The Lost Model Finder receiver uses an old faithful, the LM3914 LED bar graph indicator. The 433 MHz (315 MHz) receiver (RX) module gets plugged onto the MOD1 connector.



PICAXE PE interface

At some point you'll need to burn the plane ident program to the PICAXE chip. In case you are using an FTDI TTL-to-RS232 adapter to interface with the TXD/RXD pins of the PICAXE in combination with the PICAXE Programming Editor, you need an additional inverter circuit as shown in **Figure 3**.

Receiver

The receiver is based on a complementary module to the transmitter. However a little more care is needed in the selection of the receiver, to ensure it supplies the right signals to our circuit.
Referring to **Figure 4**, the first output of the receiver module is from the DATA pin, 14. This passes to a small-signal FET T1 and the output goes to a piezoelectric transducer Bz1 (not a piezoelectric buzzer). What you will hear from this transducer (hopefully!) is the tones and pauses generated by the transmitter. In use you listen for your personal 'call sign' that was embedded in the transmitter.
The second output from the transmitter is the RSSA output (pin 13). Basically this is a voltage that is proportional to the signal strength and used as part of the automatic gain control (AGC) in these receivers. This signal is applied to the input of an LM3914 LED driver. The RSSA signal on the prototype was found to vary between 0.4 volts and 2 volts, so the upper and lower limits of the LED driver have been set to those values. I won't go into the design of the circuit around the LM3914, since the IC has been used a zillion times in DIY electronics projects over the past decades. If you want a more detailed tutorial then point your web browser to [1] and enjoy Dave's EEVBlog #204. Say Hello to Dave for us.
The receiver is powered from four AA or AAA batteries, rechargeable or dry, although three dries should also work. Despite the transmitter's datasheet claiming a maximum voltage of 5 volts, the module has been tested to work up right up to 7 volts.
The only other major item associated with the receiver is a 4-element 'Yagi' directional antenna. This antenna was chosen for its simplicity and its directional properties, a feature which will becomes obviously necessary in the how to use section.

Building

The **transmitter** is built on a double-sided PCB using SMD components (**Figure 5**). It uses 0.1'' pitch pinheaders for all external connections except possibly K3. The artwork may be downloaded free from [2]. The PICAXE may be soldered in but also mounted in a DIL-8 socket. Only solder in the two transistors if using the battery backup version. Do not connect the RF module yet to the footprint marked K3.

At this stage you can test the transmitter driver by connecting a piezoelectric transmitter between the data and ground pins that go to the RF module. After applying power you should be able to hear your callsign.

The RF module may be connected to the board with a short length of 4-way flatcable and using pinheaders and mating IDC sockets. Alternatively, it may lie flat over the microcontroller board as shown in **Figure 6** (433 MHz version built by Elektor Labs). Make sure the solder sides of the board do not touch.

The final part is the installation of a quarter-wave antenna. A piece of stiff wire about 173 mm in length will do (I use one core of an Ethernet cable). When using a 315 MHz TX module the antenna length becomes 240 mm — remember, from $(300/f)\div 4$.

The microcontroller board, RF module and backup battery (if used) may be wrapped in a short length of shrink wrap tube.

The **receiver** PCB design is also shown in **Figure 5**. This is a single sided board for TH components so it should be plain sailing. Insert and solder all the parts into the PCB. The radio module is mounted vertically. The board and battery holder for three (or four) AA(A) cells may be mounted on the Yagi boom behind the reflector. The radiator (dipole) elements of the Yagi antenna should be connected to the input of the radio module with a length of thin 50-Ω coax cable like RG174/U or /CU. Keep the length to an absolute minimum to prevent excessive losses. Resist the temptation to use screened audio cable.

The (optional) **charger** is built on the board shown in **Figure 7**. Do not forget to fit the jumpers according to your requirements.

The **4-element antenna** is all-homebrew. Various designs are around on the Internet, and mechanically skilled members of your local model building club may be persuaded to produce a few sophisticated antennas for members. The author made the radials from coat hanger wire, although any stiff wire will do, like electrical installation wire. Try to cut the radials to the lengths shown in the drawing in **Figure 8** (dimensions for 433 MHz). The author's prototype was made from some 3-mm (1/8 in.) corrugated plastic sheet (corflute; corriboard; Polyflute), with the radials inserted into the cores. The spacing does not work out exactly, but works well in practice. Elektor Labs built their Yagi antenna from a 570 x 53 x 12 mm piece of wood and lengths of 2.5 mm²

COMPONENT LIST

Transmitter

Resistors
R1 = 10kΩ 1%, SMD0805
R2 = 22kΩ 1%, SMD0805

Semiconductors
IC1 = PICAXE-08M2, programmed
T1,T2 = DTB123YK

Miscellaneous
K1,K2,K5 = 3-pin pinheader, 0.1'' pitch
K3 = 4-pin pinheader, 0.1'' pitch (optional, see text)
K4,BT1, 2-pin pinheader, 0.1'' pitch
Transmitter module (on K3), ASK, 433MHz ISM band, type approved, e.g. Quasar Electronics type QAM-TX1 (433 MHz), Farnell/Newark # 1304024. USA readers, use 315MHz equivalent
LiPo battery, 3.7V, 130mAh
PCB # 120139-1

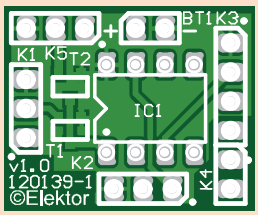
Receiver

Resistors
R1 = 1kΩ
R2 = 620Ω 1%
R3 = 360Ω 1%
R4 = 2.7kΩ

Capacitor
C1 = 100nF, 5mm or 7.5mm pitch

Semiconductors
D1–D10 = LED, orange, 2.5x5mm rectangular, 20mA
T1 = 2N7000
IC1 = LM3914

Miscellaneous
BZ1 = (not on board) piezo transducer, flying leads, Farnell/Newark # 1193640
K1,K2,(BZ1) = 2-pin pinheader, 0.1'' pitch
(MOD1) = 15-pin pinheader, straight, SIL, 0.1'' pitch
MOD1 = AM SuperHet Receiver, QAM-RX3 (433MHz), RS Components # 742-4484. USA readers, use 315MHz equivalent
Battery holder for 3 or 4 AA(A) batteries, see text
PCB # 120139-2



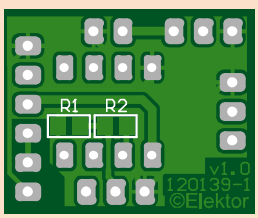
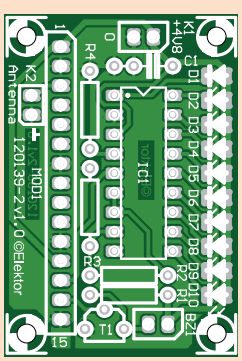


Figure 5. The Transmitter board is double-sided and takes SMD parts; the Receiver board is a single-sided design for through-hole parts. TX board shown at 150 % of true size.



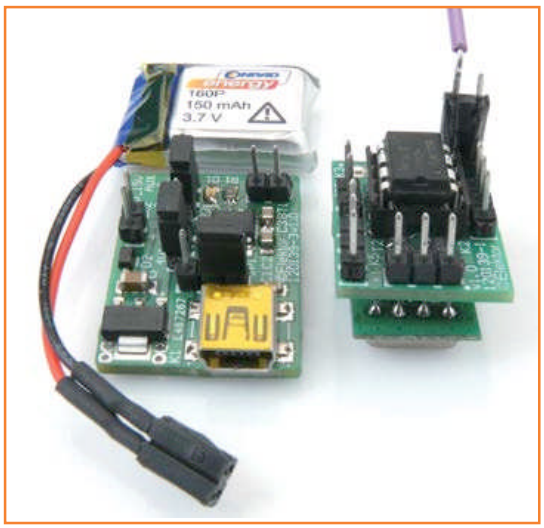


Figure 6.
LiPo Charger board (left) and TX assembly (right). The PICAXE board and the transmitter module may be connected without a cable, simply by bending and soldering the four pins straight to K3.

csa electrical installation wire (**introductory photograph**) fitted at the exact positions from Figure 8. The boom is long and wide enough to hold the RX board and the battery holder. This antenna should have about 7 dB gain, allowing the TX to be found at a distance of about 1,000 feet (300 m) in a built-up area as Labs were able to measure in and around their location at Elektor House.
Connect the dipole elements of the antenna to the receiver via the previously installed cable. Sticklers for nanovolts may want to add a balun for proper matching of the symmetrical radiator to the asymmetrical coax cable, but in this case the work and cost may defeat the purpose.

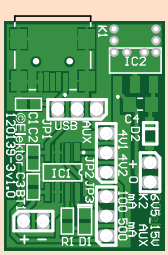


Figure 7. The charger board is a double-sided design for surface mount components.

COMPONENT LIST

LiPo Charger (optional)

Resistor
R1 = 1.5kΩ SMD 0805

Capacitors
C1 = 100nF SMD0805 X7R
C2,C3 = 10µF 10V, SMD 0805, X7R
C4 = 10µF 25V, SMD 1206, Y5V

Semiconductors
D1 = LED, red, SMD 0805
D2 = PMEG2010AEH, Farnell/Newark # 1510673
IC1 = MAX1811ESA+, Farnell/Newark # 1593327
IC2 = NCP1117ST50T3G, Farnell/Newark # 2112617

Miscellaneous
K1 = mini USB receptacle type B, SMD
K2,(BT2) = 2-pin pinheader, 0.1'' pitch
JP1,JP2,JP3 = 3-pin pinheader, 0.1'' pitch, with jumper
PCB # 120139-3

Testing

Fortunately there is no calibration required. Leave the transmitter off at this stage. Turn on the receiver and you should hear white noise with the occasional pop. The LED indicator should be showing the lowest LED on.
Turn on the transmitter. You should now hear your callsign, and the bar graph should be moving up and down in synchronization with the tones of the call sign. The bar graph should go to full scale when the transmitter and receiver are within 10 feet or so (3 m) of each other.

In use

It's going to take a little practice in order to successfully use the Lost Model Finder. It's best to initially have a friend hide the transmitter in a park or garden and then try and find the transmitter.
The antenna is directional and the 'sharp' end is the end with the greatest sensitivity. When looking for the transmitter, hold the antenna in front of you and rotate a full circle. Listen for your call sign to make sure you're chasing *your* transmitter. The bar graph will indicate where the maximum signal is. Walk in the direction of maximum. Continue doing this, stopping occasionally to check your bearings. You should see that the bar graph increases.
At some point you will get close to the transmitter and the bar graph will be at maximum. Now point the insensitive ('blunt') end of the antenna in the direction where you think the transmitter is. Now when you rotate the antenna around you, you are not looking for the maximum signal but the minimum signal. You're sort of sneaking up from behind.
You can also change whether you hold the antenna radials parallel to the ground, or perpendicular, or any other orientation.
The last few ten feet or so are the hardest in finding a plane, especially one lost in vegetation or scrub. Also don't forget to look up to the tree top canopy! You never know where these models are hiding.

(120139)

[1] Dave's EEVBlog # 204:
www.youtube.com/watch?v=iIKGvHjDQHs&feature=player_embedded)

[2] Project page:
www.elektor-magazine.com/120139

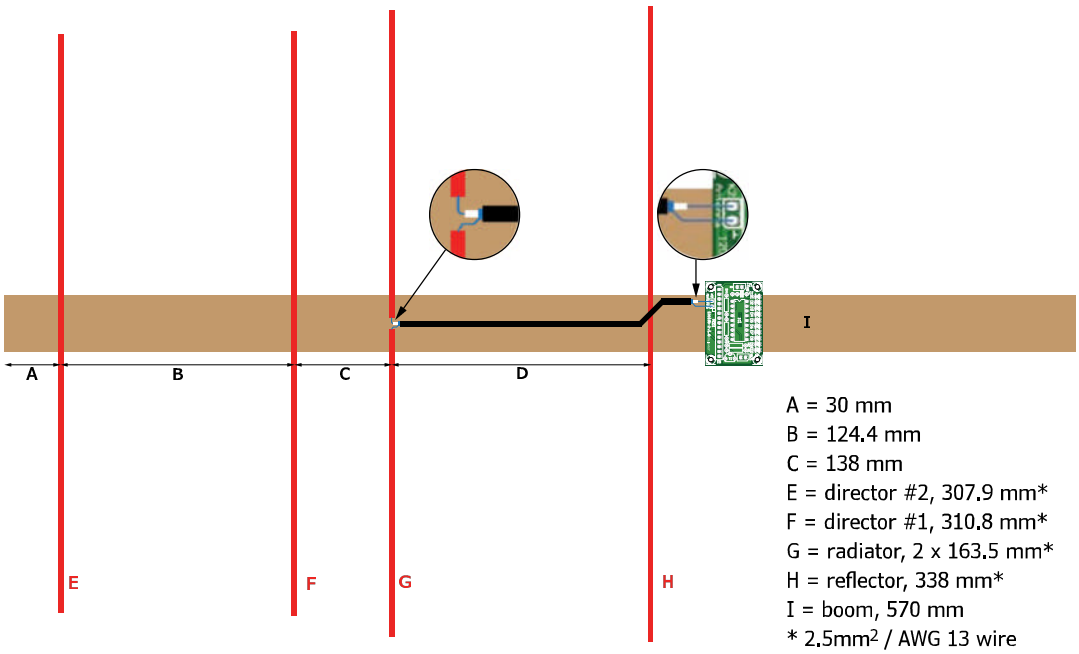
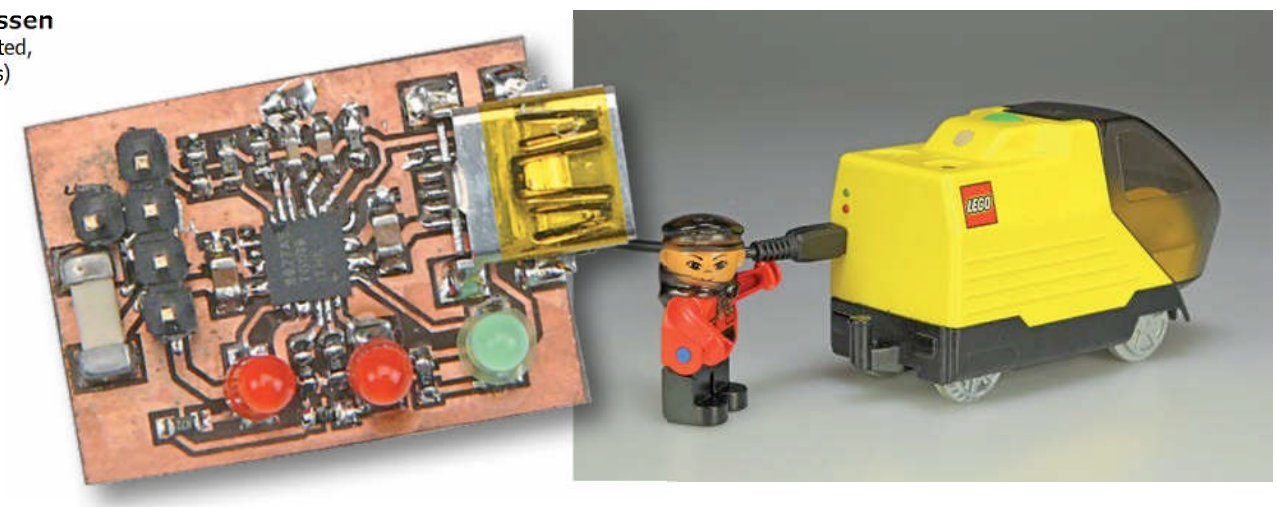


Figure 8.
Experimental Yagi antenna with the receiver electronics attached on top of the wooden 'boom', behind the reflector. The directors, radiator and reflector were made from electrical installation wire (c.s.a. around 2.5 mm² / 13AWG) and secured to the wooden boom with tie wraps — after carefully centering on the boom, of course. For safety, the radial ends should be covered with a blunt object or some putty.
Gain is approximately 7 dB. For 315 MHz use, the element spacing and lengths should be scaled up by a factor 1.37.

Lithium-ion Battery Recycling Made Easy

meaning: without a microcontroller

By **Fons Janssen**
(Maxim Integrated,
The Netherlands)



It can be quite tricky to reuse Lithium-ion batteries from discarded equipment since these cells are often charged inside the equipment, meaning that there won't be a separate charger that can also be reused. Luckily, it turns out to be fairly easy to build a charger for used (or new) Li-ion cells.

Like most people, you probably have some old equipment lying around that uses a Li-ion battery for the supply. This type of battery has been used in most portable equipment produced in the last few years because it can be easily made in various sizes and shapes, whilst it also has a relatively large capacity (compared to NiMH and NiCd).
What can you do with that old MP3 player or mobile that's been replaced by a newer, better version? The electronics can usually not be used for any other purpose, but that battery can still come in useful, even if it's for use in a toy. Since electronics hobbyists tend to be an inventive lot they will usually find a way to incorporate and connect a recycled battery. The author, for example, has built a Li-ion battery into a Lego train to replace three penlight cells (see the main photo). This still leaves the need to charge the battery.

The original equipment usually contains a special charger circuit for the battery, most likely on a small part of the PCB in the equipment. It is difficult to figure out which components are part of the charger circuit since no circuit diagram is normally made available for portable equipment. In that case we'll just have to build our own Li-ion charger!

The circuit

The charger circuit described in this article is built around a Li-ion charger IC made by Maxim (MAX8677A, see the block diagram in **Figure 1**). This IC works completely autonomously so that there is no need for a microcontroller (and hence no software!). A number of LEDs is used by the IC to indicate the state of the charging process. The IC is very flexible and contains a 'smart

power selector' (see **Figure 2**), which consists of three electronic switches that direct the charge and load currents according to the situation. With an external power source, the IC can use the available power to both charge the battery and supply the load. Should the load require more power than the charger can deliver then the IC can make the battery supply the extra current. When there is no external power source available the load is obviously powered solely by the battery.

The IC can be powered from a USB port via pins 15 and 16 (USB). In this case the current drawn is limited to 500 mA (the maximum for a USB2.0 port). The IC can also be powered from an adapter via pins 2 and 3 (DC), where the current limit can be increased up to a maximum of 2 A. In the circuit for the charger in this article (**Figure 3**) we make use of the DC input, which gives us greater flexibility in setting the various limits. The operating voltage at the input is between 4.1 to 6.6 V. If the voltage becomes too high, the IC turns off the input to prevent it from overheating. The IC can survive voltage spikes up to a maximum of 14 V.

The charging status is provided by D1, D2 and D3. The following three states are indicated:

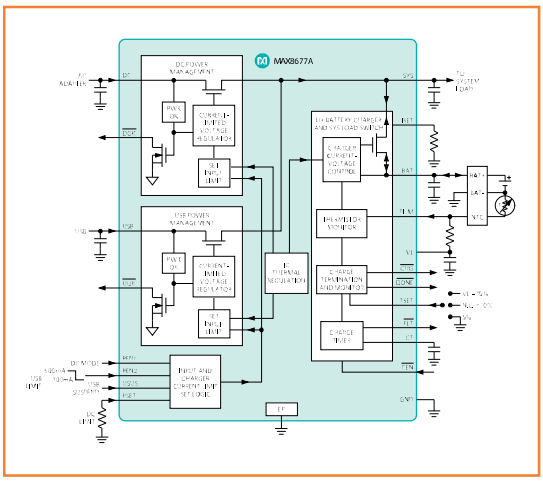


Figure 1.
Block diagram of the internal circuit of the MAX8677A.

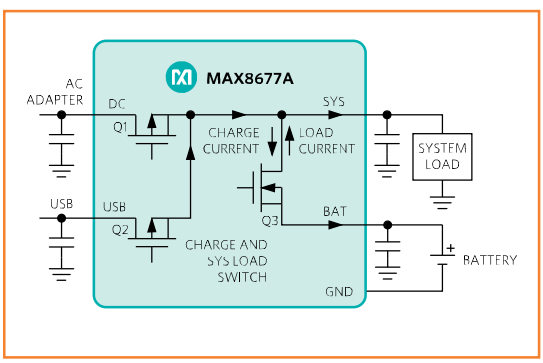


Figure 2.
The 'smart power selector' splits the charge and load currents according to the demand.

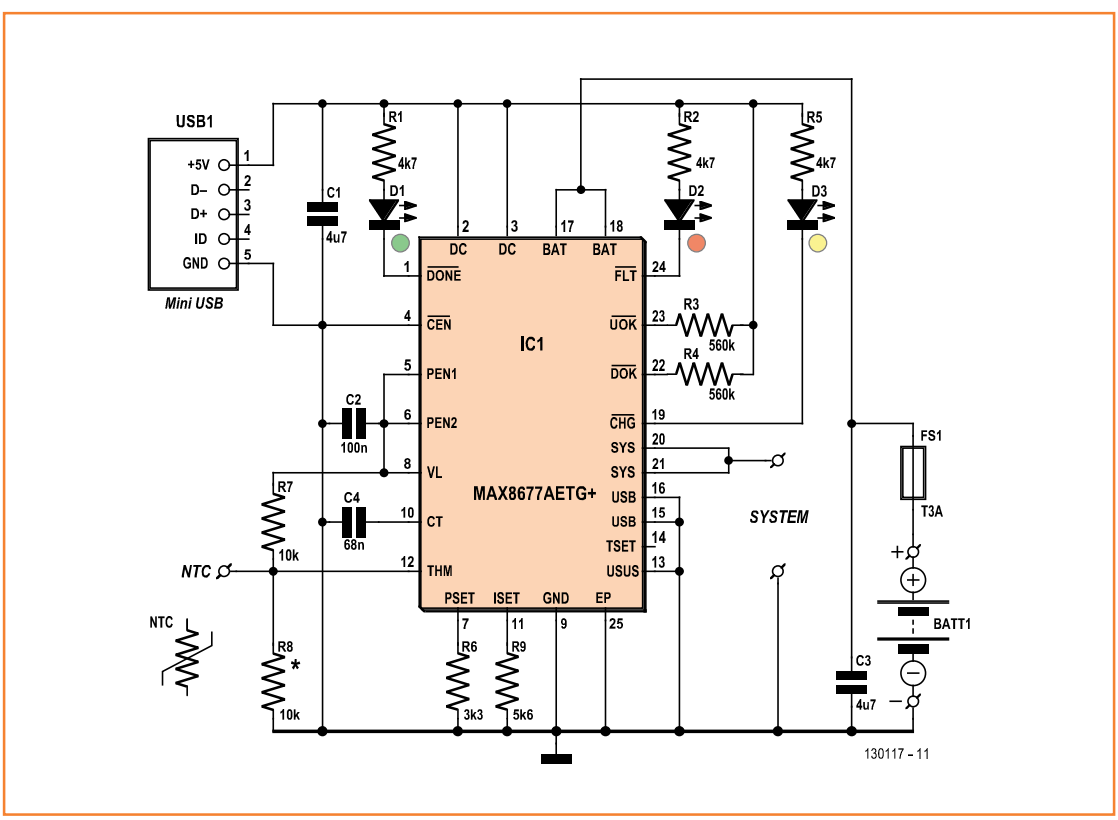


Figure 3.
The complete charger circuit consists mainly of the MAX8677 and a mini-USB connector.

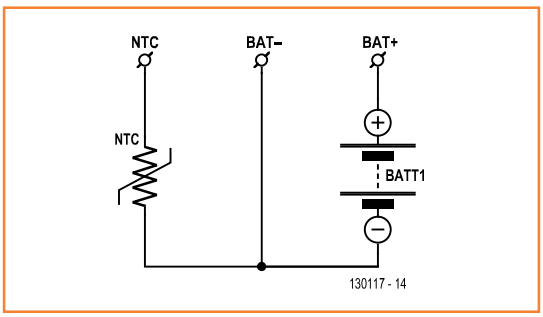


Figure 4.
Most Li-ion batteries with three connections have an internal NTC connected as shown.

The battery is being charged (LED D3), the battery is fully charged (LED D1), or the battery is faulty (LED D2).
There are two current limits that can be set with this IC: one for the maximum charge current and one for the maximum input current. It's clear that the second value should always be larger than the first. If this isn't the case, the programmed maximum charge current can never be reached as it can not go higher than the maximum input current.
Both these limits are set using a resistor:
Maximum charge current:
$I_{CHGMAX} = 3000/R_{ISET} = 3000/R9 = 3000/5.6k\ \Omega = 535$ mA
Maximum input current:
$I_{DCMAX} = 3000/R_{PSET} = 3000/R6 = 3000/3.3k\ \Omega = 909$ mA

You can obviously select more suitable values depending on the power rating of the adapter, the power consumption of the device and the desired load current. The IC can provide a maximum charge current of 1.5 A.
We've used a mini-USB connector, which makes it easy to power the circuit with contemporary mains adapters. This also ensures that we're using a 5 V supply. The maximum input current should be adjusted according to the rating of the adapter. The circuit will function just fine when the adapter can provide a current of at least 1 A.

Component list

Resistors
(default: SMD0603)
R1,R2,R5 = 4.7kΩ
R3,R4 = 560kΩ
R6 = 3.3kΩ
R7 = 10kΩ
R8 = 10kΩ (only if NTC in battery)
R9 = 5.6kΩ

Caoacitors
(default: SMD0603)
C1,C3 = 4.7µF (SMD0805)
C2 = 100nF
C4 = 68nF

Semiconductors
D1 = LED, green, 3mm
D2 = LED, red, 3mm
D3 = LED. yellow, 3mm
IC1 = MAX8677AETG+ (24-pin TQFN)

Miscellanous
USB1 = mini USB connector, PCB mount, SMD, (e.g. Molex 67803-8020, RS Components # 720-6618)

FS1 = fuse, SMD, rating dependent on battery (e.g. LittleFuse nanofuse 3 AT, Farnell/Newark # 1596930RL)
Print artwork # 130117-1 from [2]

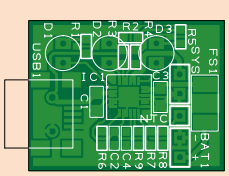


Figure 5.
The PCB designed for this charger has been kept as small as possible to make it easier to build it into existing equipment.

Is there an NTC?

Batteries are often provided with an NTC, which is used to prevent them being charged at temperatures that are too high or too low. The battery therefore has three connections: a positive terminal (BAT+), a negative terminal (BAT-) and a connection for the NTC (see **Figure 4**). You should be aware that some batteries with three connections have only a normal resistor inside that is used for identification. Its value will be constant, so won't vary with the temperature of the battery.
When an NTC is used it should be connected between the THM pin and ground (via the BAT- connection). A resistor (R7) is also connected between the THM pin and a reference voltage (V_L), which creates a potential divider. The value of the resistor is chosen such that it has the same value of the NTC at a temperature of 25 °C. The voltage at the THM pin at 25 °C will then be equal to 0.5 V_L. When the temperature rises/falls, the resistance of the NTC falls/rises as will the voltage at the THM pin. The IC will only charge when this voltage is between 0.28 V_L and 0.74 V_L. With contemporary NTCs this corresponds to a temperature between 0 °C and 50 °C. When no NTC is available you should add R8, which causes the voltage at the THM pin to be 0.5 V_L.

Connection tips

If you're reusing a cellphone battery it will already have a protection circuit as standard, which protects the battery from overloads and from being discharged too deeply. However, if you want to use a single cell, for example from a battery pack from an old laptop, then you will have to make your own protection circuit. The circuit inside the pack will have been designed to protect the whole pack and can therefore not be used to protect a single cell.

A simple fuse (shown in the circuit diagram as FS1; this is an SMD fuse on the PCB) offers sufficient protection from overloads, which means that separated cells are perfectly usable. However, a fuse doesn't offer any protection against deep discharging. When these types of cells are discharged too much it is possible to damage them. This can happen when an ohmic load is attached for too long a period, such as a small incandescent light bulb. Most devices will stop working once the supply voltage has dropped below a certain value, which stops the cell from discharging further. Whether or not a fuse offers enough protection therefore depends very much on the type of device that is connected.

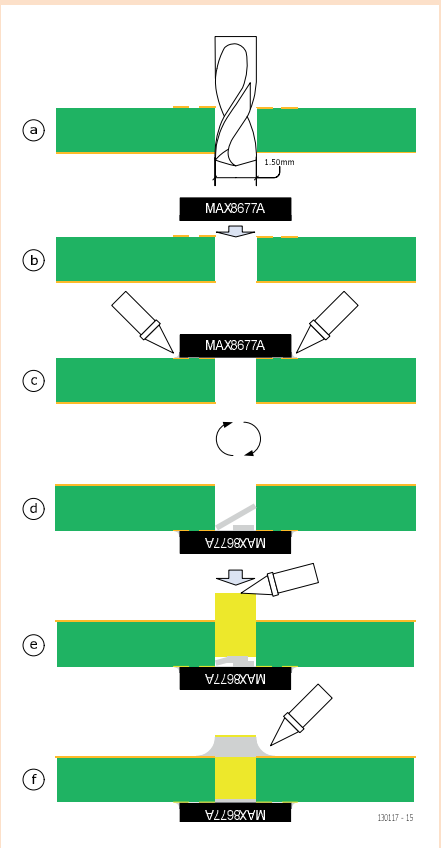
Construction

A compact PCB has been designed for this project, which uses a large number of SMDs (see **Figure 5**). This keeps the PCB very small, which makes it easier to build into the device. The board layout (made using DesignSpark) is available as a free download from [2].

You will need some dexterity and soldering experience in order to mount the SMDs. For the TQFN packaged IC you should ideally use a reflow oven, since the pins and the exposed pad are on the underside of the 4x4 mm package. However, the author has thought of a method that can be used to solder the IC by hand, which is described in detail in the inset. There are holes on the PCB for the connections to the LEDs, battery and the load, so they can be connected easily via wires. The mini USB connector has two plastic pins that fit in the associated holes on the PCB, providing alignment of the socket. If you don't want to use the mini USB connector you can use these two holes for connecting the power. In this case you have to make sure that the supply voltage is 5 V.

(130117)

Soldering a TQFN by hand



Mounting the IC with a hot air soldering iron is possible if you're experienced enough, although a reflow oven makes life much easier. The method described here shows how to use an ordinary soldering iron to mount the IC, even though the home made PCB is not through-plated.

Locate one hole in the center of the IC's exposed pad instead of 9 like in the PCB design shown. Drill this hole using a drill bit with a diameter of 1.5 mm (a). Position the chip on the PCB (b) and solder all contacts along the sides of the chip (c). Use Litze wire to tidy everything up again (the author used a stereoscopic microscope to get a good view of everything). When all contacts on the top side have been soldered properly we turn over the board and drop a few pieces of solder into the hole (d). Find a piece of solid copper cable that fits snugly in the 1.5 mm hole (one of the cores in a solid core mains cable would be a good choice) and use a file to make one of the ends completely flat. Put this end through the hole and heat it up using the soldering iron (e). At some point the piece of copper wire gets so hot that the pieces of solder in the hole start to melt. The copper wire will then drop down slightly and makes contact with the exposed pad of the chip. Then solder the piece of wire to the ground plane on the solder side of the PCB (f).

You will now have a good electrical and thermal connection between the exposed pad of the chip and the ground plane on the solder side of the PCB.

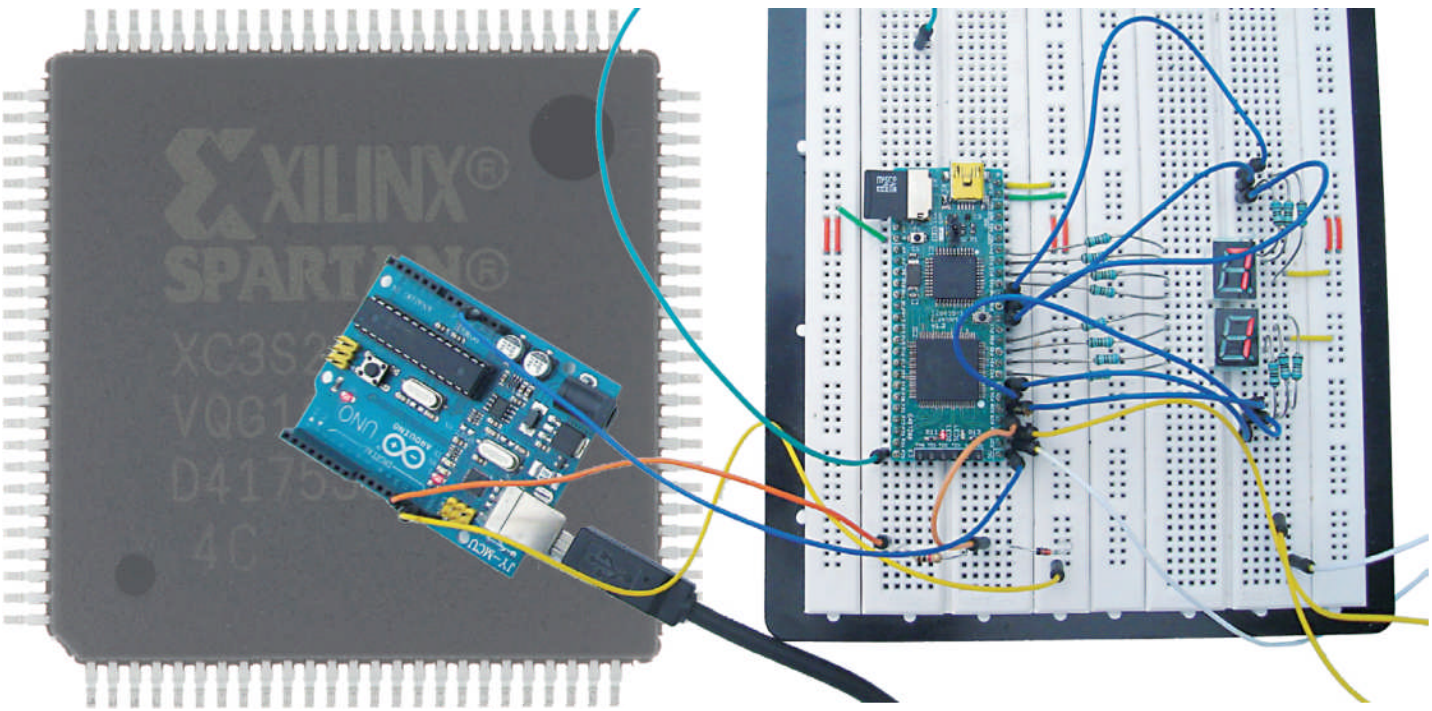
Internet Links

[1] http://datasheets.maximintegrated.com/en/ds/MAX8677A.pdf

[2] www.elektor.com/130117

Taming the Beast (5)

Programming an array of 250 Kgates



By **Clemens Valens**
(Elektor.Labs)

Although it's perfectly possible to design FPGA applications using schematic diagrams with logic symbols, in practice this is usually done with a hardware description language. An advantage of the latter approach is that complex functions are often easier to express in algorithms than in schematics. Accordingly, in this installment we guide you through the process of programming an FPGA application.

In the previous installment [4] we took a cautious initial look at the hardware description languages VHDL and Verilog for testing an application. In this final installment we delve into hardware description languages more extensively – in this case, to define an application. However, working with two languages is too difficult, so I had to make a choice. After many sleepless nights and consulting dozens of experts, I ultimately decided to cast my lot with VHDL. There are various reasons that can be given for this choice. One is that VHDL should require less simulation time because it is more difficult to arrive at a synthesizable design. That might sound like a reason for choosing Verilog instead, but it reflects the fact that simulation often takes longer than synthesis, so in the end you're done faster. To draw an analogy with computer programming (Verilog fans should look the other way and plug their ears at this point), instead of just throwing together a bit of code and then using the debugger to turn it into a working algorithm, you can first spend some time thinking about your algorithm and then use the debugger to make sure it works properly.

The application I have chosen here is a DCF77 decoder. I can hear the snide remarks already, but there are good reasons for this choice. The radio signal from the DCF77 atomic clock broad-

cast station near Frankfurt in Germany can be received on 77.5 kHz easily in virtually all of Europe with an inexpensive module. The signal coding is fairly simple – it consists of a sequence of 59 pulses (one per second) which can have two different lengths (100 ms for "0" or 200 ms for "1"), which collectively represent the bits of the time code. The end of the code sequence is marked by omitting the final pulse (number 60, corresponding to bit 59). A new code is sent every minute. The date and time are in BCD format, and there are several check bits that can be used to verify the validity of the data. This signal is a good starting point for a project with VHDL (or Verilog) because it is relatively easy to show the results on a 7-segment display with simple logic. For those of you who do not have a DCF77 receiver or who cannot receive the signal because they live too far away from Frankfurt, we have written a DCF77 simulator program for Arduino [5]. That way everyone can join in.

Let's start with a functional design for a DCF77 decoder. Here we take the 'naïve' approach, which means we assume that the input signal will usually be good and will stay nicely within specifications. Thanks to this assumption, the design can be fairly simple.

As previously mentioned, the input signal consists of pulses with lengths of 100 ms and 200 ms, which represent zeros and ones respectively. If you sample the input signal at a point 150 ms after the rising edge of each pulse, you will thus see either a '0' or a '1' (**Figure 1**). If you also measure the time between two successive rising edges, you can detect the missing pulse and therefore identify the start of the time code. The bits that are found in this manner are fed into a shift register. Then the BCD sequences that encode the various date and time units are extracted from the shift bit string in the shift register and presented on the 7-segment display. In the C-based pseudocode, this looks roughly as follows:

```
do for each clock pulse
{
  counter = counter + 1
  if (counter == 150ms)
  {
    shift_register = (shift_register << 1) + input_signal;
  }
  if (rising_edge(input_signal) == true)
  {
    if (counter >= 1750ms)
    {
      show_content(shift_register);
    }
    counter =;
  }
}
```

The counter is incremented on each clock pulse. When the counter reaches 150 ms, the input signal is sampled and the sample is fed to the shift register. When a rising edge is detected in the input signal, a test is made to see whether the counter has exceeded the maximum value. If it has, the shift register is full and the data can be displayed. After this the counter is reset to 0. That's all there is to it. You could add all sorts of bells and whistles to this, such as error detection and more robustness, but that falls outside the scope of this introductory example.

How do you do this in VHDL?

Doing the same thing in VHDL is fairly easy because translating pseudocode into VHDL is straightforward. The result is the following code fragment:

```
1  process (clock) is
2  begin
3    if rising_edge(clock) then
4      counter <= counter + 1;
5      if (counter=t150ms) then
```

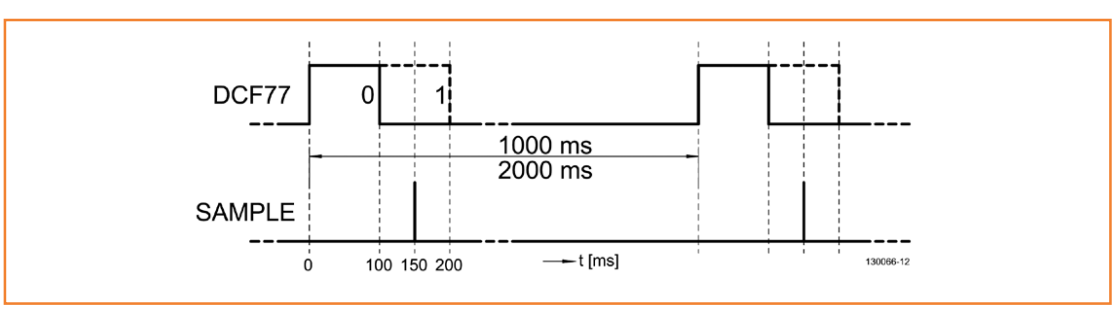


Figure 1.
The DCF77 signal and sampling points.

```
 6       bits <= input & bits(58 downto 1);
 7     end if;
 8     if input_rise='1' then
 9       if (counter>=t1750ms) then
10         data <= bits;
11       end if;
12       counter <= 0;
13     end if;
14   end if;
15 end process;
```

The definitions and a bit of VHDL syntax are missing here, but the algorithm is there. I added the line numbers here because they make it easier to explain how the code works. In this code, `counter` is self-explanatory and `bits` are the stages of a 59-bit shift register. Let's start at the top with line 1.

This is a process, which means that it will be executed by the FPGA. Nothing happens without a process. There can be multiple processes, and they are all executed at the same time. Within a process, the order of execution is normally top to bottom. Our process is dependent on the signal `clock`, which means that the process is only executed when the value of `clock` changes. The process starts after line 2 and runs until line 15. Line 3 causes everything in the process to be synchronized to the rising edge of the clock signal. This is made possible by the function `rising_edge`. You will often see the following construction in VHDL code:

```
if clock'event and clock='1' then
 ..
end if;
```

This does the same thing as `rising_edge`, but it is a bit old-fashioned. Here we use `rising_edge`. There is also a function called `falling_edge`.
The counter is incremented by 1 in line 4. Easy, isn't it? Yes, but you should be careful because whenever you use the increment function, which is represented by the "+" sign, you have to include the `numeric_std` library in the list of libraries to be used (as described elsewhere in this article).

In line 5 the counter value is compared to a constant that corresponds to an interval of 150 ms with a clock frequency of 8 MHz (the FPGA clock rate). If the counter is at 150 ms, the current value of the input signal is fed into the `bits` shift register in line 6. This is done from the left end of the shift register, unlike the pseudocode where it is done from the right. We chose this approach because the time code starts with the least significant bit, so the bits are fed into the shift register in the right order for our purpose. After 59 pulses, bit 0 is at position 0. The shift operation looks a bit strange because the concatenation function "&" is used for this purpose. This function simply joins the left part of the expression to the right part. Here the right part consists of the 58 bits from 1 to 58 (in reverse order, without bit 0), which are shifted to positions 57 to 0. The left part is the 1-bit input signal, which now ends up in bit 58. The details of what ISE (or actually XST) does with this construction are not so important here; what matters is that it works (and it does).
Line 8 checks to see whether a rising edge has been detected in the input signal. Exactly how that works is not obvious from the code fragment (I'll explain this later), but if a rising edge has been detected, line 9 checks whether the counter has exceeded the maximum value of 1750 ms. If it has, the content of the shift register is copied to a data register for further processing, which is done in another fragment described below.
Line 12 resets the counter to zero. It is not necessary to flush the shift register, since it is completely filled every minute.

The lines not specifically mentioned here are only there to conform to VHDL syntax rules.
To complete this fragment, all you need is an edge detector, some VHDL syntax glue, and the specifications of the input and output signals for this process. In the final design I also added a couple of signals that blink the LEDs on the FPGA board so you can see whether something is actually happening inside the FPGA.

Edge detection

The edge detector is a key part of this design. At first I used a simple scheme that compared the current value of the input signal with the value during the previous clock pulse. This was not reliable and caused problems, with the result that synchronization was occasionally lost. The solution turned out to be adding another flip-flop so that instead of comparing the current value with the previous value, we compare the previous value with the one before it. In other words,

instead of comparing t = n with t = n - 1, we compare t = n - 1 with t = n - 2.
You may be asking yourself whether the function `rising_edge` could have been used for this purpose, but unfortunately the answer is that this won't work. If you try this, the XST synthesizer will think that a clock signal is involved, but since this isn't the case, XST doesn't know what to do and therefore generates an error message.
I included the edge detector in the project as a separate module (a function in VHDL; see **Listing 1**), but this is not mandatory. However, it has the advantage that you can easily use the function in another project.
The module starts with a **library** instruction, followed by a **use** instruction. The purpose of these instructions is to ensure that the module can access the appropriate standard functions and signals. If necessary, you can also add other libraries here, such as `numeric_std` (for "+", as you may recall). Unlike most programming languages, these instructions do not apply to the entire file in which they appear, but only to the first **entity** block and associated **architecture** block(s) appearing after the instructions. This means that each **entity** block is normally preceded by library statements.

Listing 1. An edge detector module in VHDL.

```
library ieee;
use ieee.std_logic_1164.all;

entity edge_detector is
  port (input : in std_logic;
        clock : in std_logic;
        rise : out std_logic);
end edge_detector;

architecture behavioral of edge_detector is
begin
  process (clock)
    variable history : std_logic_vector(1 to 3);
  begin
    if rising_edge(clock) then
      rise <= history(2) and not history(3);
      history := input & history(1 to 2);
    end if;
  end process;
end behavioral;
```

Now let's have look at the **entity** block. It can be regarded as essentially equivalent to a symbol in a schematic diagram. This is where the inputs and outputs of the component are defined in the **port** section. Here you see that all the signals are of type `std_logic` from the `std_logic_1164` library, which means that they are logic signals that can assume various states defined in the library, including "0", "1", and quite a few others. The signals with the suffix **in** are the inputs of the module, while the signals with the suffix **out** are the outputs.
After the **entity** block there is an **architecture** block that defines the function. The latter block has a name (`behavioral`) and is an implementation of the specified entity (in this case `edge_detector`). The name `behavioral` is assigned when you let ISE create a VHDL module. Another frequently seen name is `rtl`, which stands for Register Transfer Level (the level where VHDL code is compiled), but you can also choose your own name.
The edge detector is implemented as a **process** that is dependent on the signal `clock`. The instructions in the process are executed each time the value of this signal changes.
After the process declaration, there is a list of variables that are needed inside the process but not outside. Here there is only one: `history`, a 3-bit shift register. This variable is a vector, which means that it consists of several bits. Note that

Listing 2. DCF77 timecode decoder with edge detector.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dcf77_decoder is
    port ( input : in std_logic;
           clock : in std_logic;
           data : out std_logic_vector (58 downto 0) );
end dcf77_decoder;

architecture behavioral of dcf77_decoder is

  component edge_detector is
    port (input : in std_logic;
          clock : in std_logic;
          rise : out std_logic);
  end component edge_detector;

  constant t1750ms: integer := 14000000; -- 1750 ms @ 8 MHz
  constant t150ms: integer := 1200000; -- 150 ms @ 8 MHz
  signal counter : integer := 0;
  signal bits : std_logic_vector(58 downto 0) := (others => '0');
  signal input_rise : std_logic := '0';

begin
  edge_detect: edge_detector port map (input => input,
                                       clock => clock,
                                       rise => input_rise);

  process (clock) is
  begin
    if rising_edge(clock) then
      counter <= counter + 1;
      if (counter=t150ms) then
        bits <= input & bits(58 downto 1);
      end if;
      if input_rise='1' then
        if (counter>=t1750ms) then
          data <= bits; -- Transfer data.
        end if;
        counter <= 0; -- Clear counter.
      end if;
    end if;
  end process;
end behavioral;
```

here the vector is defined as `(1 to 3)`, while the shift register in the DCF77 decoder is defined as `(58 downto 0)`. Both of these definitions are valid, but you must be careful to avoid accidentally getting the bits confused when you use the two together.

The edge detector is only active after a rising edge of the clock signal. When that happens, it determines the value of the output signal `rise` from bits 2 and 3 of `history`, and then feeds the current value of the input signal into the shift register in the same way as the code fragment for the DCF77 decoder.

As you can see, each block ends with an **end** statement (or perhaps **end if**), possibly followed by the name of the block concerned, so your fingers will be busy on the keyboard. This is one of the drawbacks of VHDL: it requires a lot of typing. **Figure 2** shows what ISE thinks the edge detector should look as a schematic diagram. I personally find this interpretation somewhat strange, but maybe I'm missing something. Let's just say that ISE is not the world's best draftsman.

I discussed this module in so much detail to illustrate what a VHDL module looks like. All modules follow this pattern of **library**, **entity** and **architecture**, including the DCF77 decoder module.

Using modules

If you want to use the edge detector in the DCF77 decoder, you have to somehow make XST aware of its existence. There are various ways to do this, such as using a library, but here we use the component method. **Listing 2** shows the details of how this works. This listing also shows the complete DCF77 decoder module, including the **entity** block and the associated **architecture** block, the input and output signals, and the local variables and constants. Just have a good look at it, take your time, and use the Web for help if necessary – it's not all that difficult.

You include the edge detector in the module by declaring it as a component at the top in the **architecture** block. Then you copy the **entity** block and replace the word **entity** by **component**. The next task is to link in this block, which you do after the start of the **architecture** block by generating an instance of the block. For this purpose, you have to specify a label and then

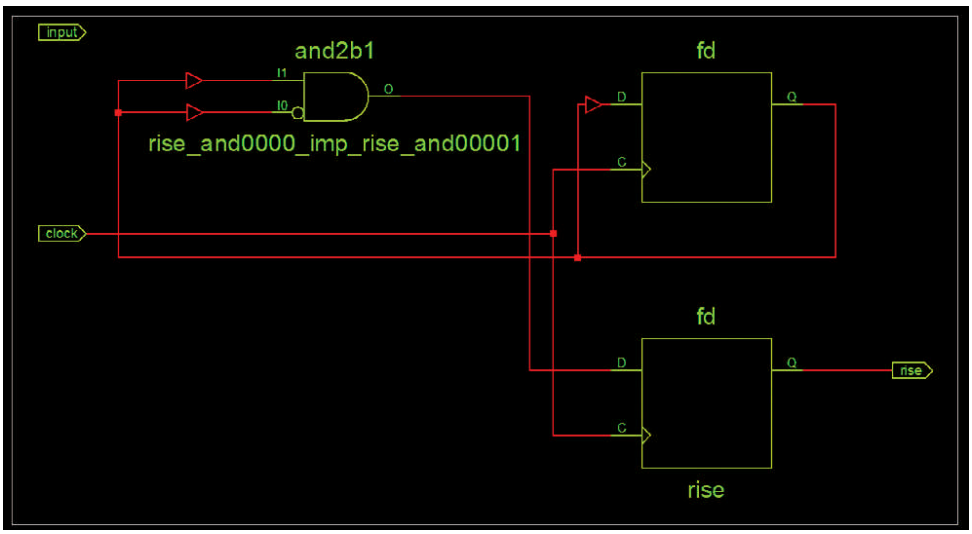


Figure 2.
This bizarre schematic is the result of letting ISE's RTL Viewer draw the schematic diagram of the edge detector described in this article (*Design* tab, *Synthesize – XST* → *View RTL Schematic*). The signal input is not connected to anything, and the operation of this circuit is a complete mystery to the author. Maybe he got something wrong?

use a **`port map`** to indicate which signals (ports) of the DCF77 decoder on the right in the folder must be linked to the signals (ports) of the edge detector on the left in the folder. The clock and input signals are already taken care of because they are inputs (ports) of the DCF77 decoder as part of the **`entity`** block. You have to add a local signal of the same type for the edge detector output signal `rise`. Here I gave it the name `input_rise`. Now you can use this signal in your process. It will assume the value "1" for one clock period when a rising edge is detected in the DCF77 signal.

If you forget to map an input signal of a component, you will see a rather baffling error message. It tries to explain that the forgotten signal does not have a default value and may therefore remain unconnected. Don't say I didn't warn you.

Display

Articles of this sort tend to be long because I try my best to explain everything properly, but

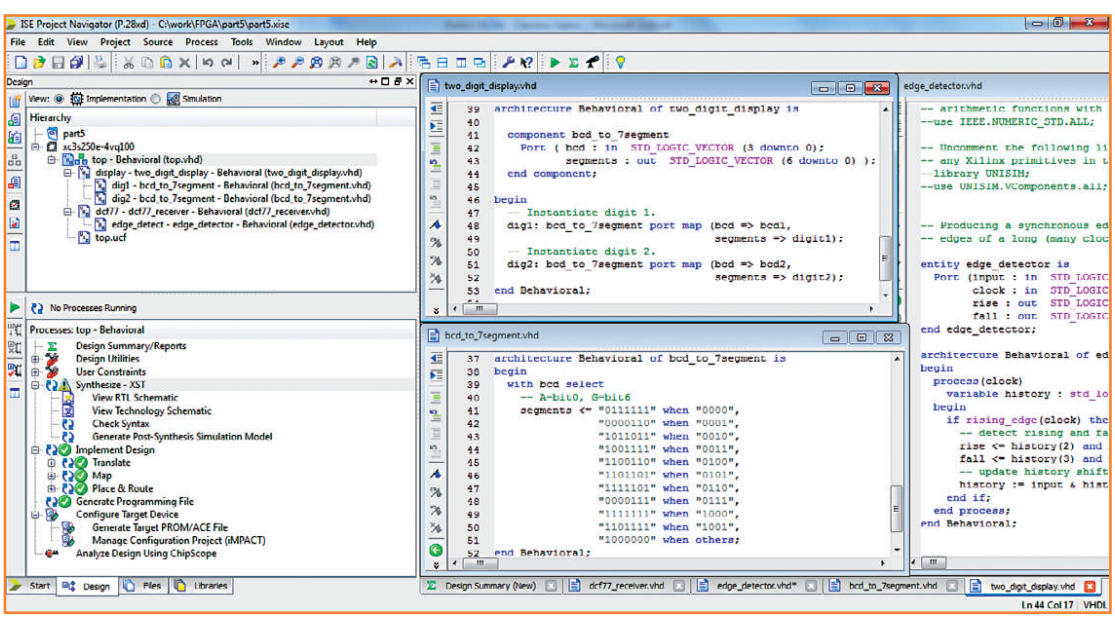


Figure 3.
An ISE screenshot showing the code for the display and the project hierarchy, along with the green check marks which prove that it's possible to compile the project and generate a bit file.

my editor is not especially fond of long articles. Nevertheless, I want to explain in an article how you can produce a working VHDL design, since otherwise the task is a bit frustrating. As I still have a lot to explain, I am omitting the code for the 7-segment display from this article. I used the same two-digit display as in the third installment [3], with a BCD to 7-segment decoder as described there. This is a standard textbook exercise, so there is no need to explain it in detail. Have a look at the project for this installment [5] to see how it all works. **Figure 3** also shows a few details.

Listing 3. The top level of the final DCF77 decoder.

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
    port ( dcf77_input : in std_logic;
           clk_in : in std_logic;
           hour_month : in std_logic;
           time_date : in std_logic;
           led1 : out std_logic;
           led2 : out std_logic;
           t_sample : out std_logic;
           digit1 : out std_logic_vector (6 downto 0);
           digit2 : out std_logic_vector (6 downto 0) );
end top;

architecture behavioral of top is

  component two_digit_display is
    port ( bcd1 : in std_logic_vector (3 downto 0);
           bcd2 : in std_logic_vector (3 downto 0);
           digit1 : out std_logic_vector (6 downto 0);
           digit2 : out std_logic_vector (6 downto 0) );
  end component;

  component dcf77_decoder is
    Port ( input : in  std_logic;
           clock : in  std_logic;
           tick : out  std_logic;
           sync : out  std_logic;
           data : out  std_logic_vector (58 downto 0) );
  end component;

  signal data : std_logic_vector (58 downto 0);
  signal bcd1 : std_logic_vector (3 downto 0);
  signal bcd2 : std_logic_vector (3 downto 0);
  signal tick : std_logic;

begin
  display: two_digit_display port map (bcd1 => bcd1,
                                       bcd2 => bcd2,
```

Back to the top

As in the previous installments, everything comes together at the top, but now the top is a VHDL module instead of a schematic diagram. This detail must be communicated to ISE in the *Design Properties*. First create a new project based on the previous one, in the same way as described at the start of the third installment [3]. Then delete all schematics, but keep the UCF file. Next, open *Design Properties* (e.g. at the bottom of the *Project* menu) and set *Top-Level Source Type* to "HDL" and *Preferred Language* to "VHDL" (if this isn't already done).

Now you can start adding new source files. To do this, open the *Project* menu or right-click the *Design* tab and select *New Source...*. Then select *VHDL Module*, enter the file name (e.g. `top`), verify that *Add to project* is ticked, and click *Next*.

Now you will see a form that you can fill in if you know which input and output signals you need. If you don't know, leave the form blank. When you're done, click *Next* and then *Finish*. ISE will create a file for you, containing the template where you have to add your VHDL code. Add as

```
                                    digit1 => digit1,
                                    digit2 => digit2);

  dcf77: dcf77_decoder port map (input => dcf77_input,
                                 clock => clk_in,
                                 tick => tick,
                                 sync => led2,
                                 data => data);

  process (clk_in) is
  begin
    t_sample <= tick;
    led1 <= tick;
    if rising_edge(clk_in) then
      if time_date='1' then
        -- Show time.
        if hour_month='1' then
          -- Show hours.
          bcd1 <= data(32 downto 29);
          bcd2 <= "00" & data(34 downto 33);
        else
          -- Show minutes.
          bcd1 <= data(24 downto 21);
          bcd2 <= "0" & data(27 downto 25);
        end if;
      else
        -- Show date.
        if hour_month='1' then
          -- Show month.
          bcd1 <= data(48 downto 45);
          bcd2 <= "000" & data(49 downto 49);
        else
          -- Show day of month.
          bcd1 <= data(39 downto 36);
          bcd2 <= "00" & data(41 downto 40);
        end if;
      end if;
    end if;
  end process;

end behavioral;
```

many files as the number of modules you intend to generate. In the present project there are five: top, DCF77 decoder, edge detector, BCD to 7-segment decoder, and 2-digit display.

The top module is shown in **Listing 3**. The names of the input and output signals in top, which are the signals named in the **entity** block of the top module, must match the names in the UCF file, because this is how the pins of the IC are linked to the VHDL code. All of the signals in the UCF file must appear here, as otherwise ISE will complain. A new feature here is the use of vectors for the display pins. You can do this in the UCF file by using an index with the name of the vector – for example, `digit1(0)` corresponds to bit 0 of the `digit1` vector in the top module.

Two components are called in the **architecture** block of the top module: `two_digit_display` and `dcf77_decoder`, and one instance of each of them is used. With a larger display, such as six digits, you could generate and link three instances of `two_digit_display`, each with its own label. For linking the two components, I defined several additional signals that are not needed outside the top module. For example, the `tick` signal is used to blink LED1 each time a bit is received. Since this LED is not connected to a pin on the FPGA board, it is also linked to the `t_sample` signal, which in turn is linked to pin P86 in the UCF file. This makes it easy to view the sampling points on an oscilloscope relative to the DCF77 input signal.

The process of the top module does not do anything especially remarkable. It is actually a mul-

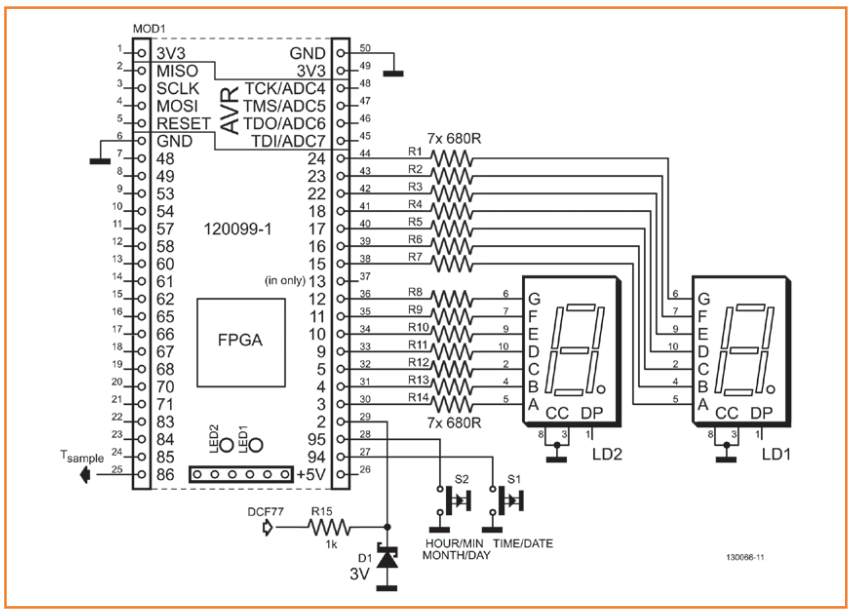


Figure 4. The schematic of the DCF77 decoder is virtually identical to the one in the third installment [3]. Since the FPGA is not especially comfortable with 5-V signals, a Zener diode is used to limit the input signal level to approximately 3 V.

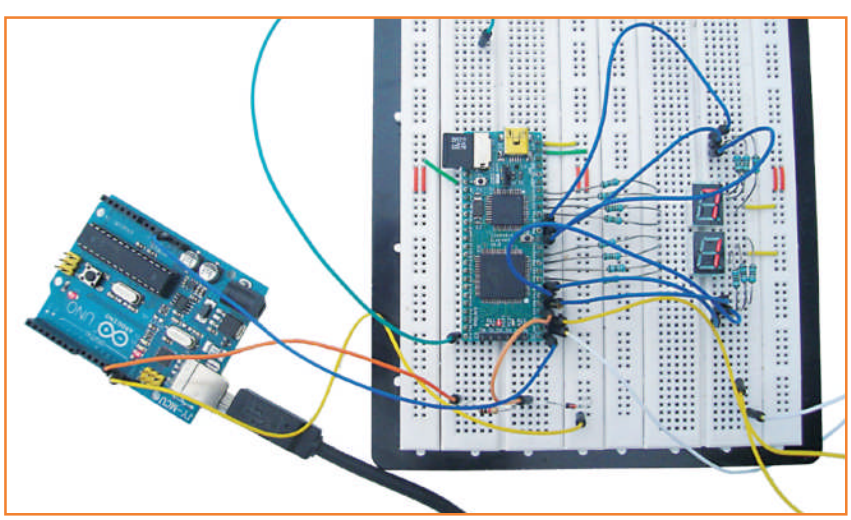


Figure 5. The test circuit being driven by an Arduino acting as a DCF77 simulator. The display shows the simulated hours figure ('17'). The FPGA board is powered from the Arduino board.

tiplexer that shows the hours, minutes, day of the month or month number on the two-digit 7-segment display, depending on the levels of the two control signals `time_date` (P94) and `hour_month` (P95). With a but of cutting and pasting ("&"), the right bits are extracted from the shift register string to form 4-bit BCD codes, which in turn are converted into 7-segment signals by the BCD decoders. Here it should be noted that it is not necessary to explicitly route the vectors `bcd1` and `bcd2` to the display component, since this is done implicitly by the port map.

When you synthesize this design, you will see a number of warnings. They are caused by the fact that some of the bits of the DCF77 shift register are not used, and that bit 3 of the vector `bcd2` is always "0" because the design does not use eight-bit data (only the year number component encompasses eight bits, but it is not used here). You will also see that a nicely ordered hierarchy has been generated on the *Design* tab (**Figure 3**), just as when you entered the design in schematic form. Everything is therefore restored to the way it was. As already briefly mentioned, ISE has an RTL viewer that is able to portray VHDL code in schematic form (more or less; see Figure 2), which emphasizes the underlying duality: a schematic is the same as VHDL code, and VHDL code is the same as a schematic.

Despite my efforts to keep things short, it has been a long journey. I have tried to convey all the essential information, but a bit of self-reliance and determination to build this project yourself are probably still necessary. Good luck!

(130066)

Web links

[1] Part 1: www.elektor.com/120099
[2] Part 2: www.elektor.com/120630
[3] Part 3: www.elektor.com/120743
[4] Part 4: www.elektor.com/130065
[5] Part 5: www.elektor.com/130066

From BASIC to Python (2)
A field report

By
Jean-Claude Feltes
(Luxembourg)

In the first part of this series we looked at how Python differs from BASIC. We went over the installation process and got our first program running. Now we go a little further with graph plotting and Fourier synthesis. Finally with not much effort we use a graphical user interface.

The Python language is particularly suited to engineering environments. Often it is advantageous to represent data graphically. The saying that a picture is worth a thousand words is also true in the world of mathematics; an engineer can interpret so much more from the plot of a curve compared to a column of sterile mathematical values. As a rule popular programming languages already feature tried and tested library routines to ease the writing process. Python is no exception to this rule.

Plotting graphs

In Python the standard library for 2D graph plotting is 'Matplotlib'. Part of the Python philosophy is summed up in its motto 'There is only one way' suggesting that there should only really be one obvious way to achieve something in Python. This does not unfortunately apply to the modules written in the additional libraries, a discovery that has already cost me many hours work. Notably in Matplotlib there exist simple procedural interfaces and complicated object orientated interfaces. Examples given in books and on the Internet use either one method or the other so it is quite easy to get confused.
The simple interface 'pyplot' facilitates very simple programming. The program in Listing 1 produces a damped sine wave oscillation. The curve it describes is shown graphically in a display window (Figure 1).
The first line of the listing imports the interface 'pyplot' as the object 'plt'. 'Numpy' is an extension to python providing support for mathematical functions. The second line shows three functions imported from the Numpy extension.
The 'linspace' routine can be used with an interval (here from 0 to 7) divided into an equal number of parts (1,000 here) and represented as a vector (array). This enables a fast and simple calculation of the value of the function. Numpy functions can also handle vectors. Using just the line:

```
y= sin(5*x)*exp(-x)
```

Calculates all 1,000 values of the y vector.
This relieves you of the more usual 'For' loop constructs and makes the code both fast and understandable.
The command 'plt.plot(x,y)' plots the data as a curve on the graph and 'plt.show' is used to show it on the graph. To display more than one curve on the chart as shown in Listing 2 it is necessary to call the plot function again.
The graph window (Figure 1) automatically shows a toolbar which allows you to zoom and save the chart. Apart from this it also indicates the cursor coordinates. Should you want to spice up the graph with some GUI elements then it will be necessary to proceed using an object oriented approach.

Example: frequency response

As an example we can plot the transfer function of a simple RC low pass filter. The transfer function shown here is a result of an RC voltage divider network consisting of a resistor and capacitor which has a complex impedance:

$$F = 1 / (1 + j \omega R C)$$

The frequency response is given by a plot of the value of F as a function of frequency (Figure 2). This is a good example to demonstrate how Python handles complex variables.
In this program (Listing 3) the Array 'f' is calculated with frequency steps incremented logarithmically. For plotting against linearly incrementing values use 'linspace()'. An array for the complex F values is calculated along with an array of the absolute values 'Fabs'. Numpy is useful here with its vector functions. There is no requirement for a 'For' loop in the calculation and the code is both short and understandable.
Finally the plot is displayed, showing a logarithmically scaled frequency axis. For a professional looking result it is important to define the grid lines:

```
ax.grid(True, which = "both", linestyle
= "-")
```

The grid lines are defined with "both" indicating the same line style for both major and minor subdivisions. Lastly a solid line style is selected. By adding the following lines of code:

```
# plot phi = f(f)
phi = angle(F)*180.0/pi
ax2 = fig.add_subplot(212)
ax2.plot(f, phi)
ax2.grid(True, which = "both", linestyle
= "-")
ax2.set_xscale ("log")
ax2.set_xlabel("f/Hz")
ax2.set_ylabel("phi/degrees")
```

We can show the phase shift introduced by the filter.

Example: Fourier synthesis

Engineers quite partial to some mathematics will be pleased to learn that some of the more useful advanced engineering concepts such as Fourier synthesis are also catered for in Python.
The example given in Listing 4 shows how a square wave can be synthesized by summing 30 weighted harmonics of a fundamental sine wave. Figure 3 shows quite an impressive output waveform from what is a relatively short program listing. More or fewer harmonics can be used to

Listing 1: Sinewave.py

```
import matplotlib.pyplot as plt
from numpy import sin, exp, linspace

x=linspace(0.0, 7.0, 1000)
y= sin(5*x)*exp(-x)

plt.plot(x, y)
plt.show()
```

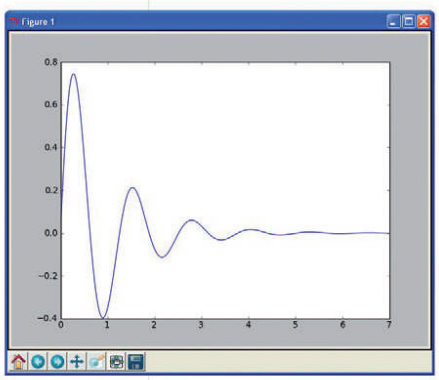


Figure 1.
The plot of a damped oscillation given in Listing 1.

Listing 2: Multigraph.py

```
mport matplotlib.pyplot as plt
from numpy import sin, exp, linspace

x=linspace(0.0, 7.0, 1000)
y1 = sin(5*x)*exp(-x)
y2 = y1* 0.5

plt.plot(x, y1)
plt.plot (x, y2)
plt.show()
```

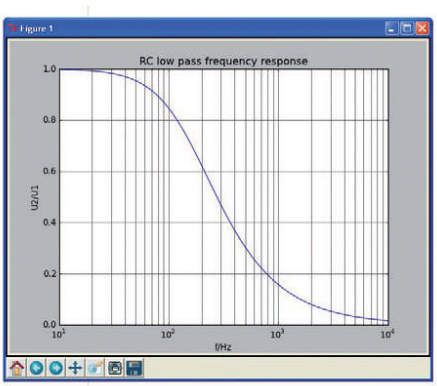


Figure 2.
The frequency response of an RC low pass filter given in Listing 3.

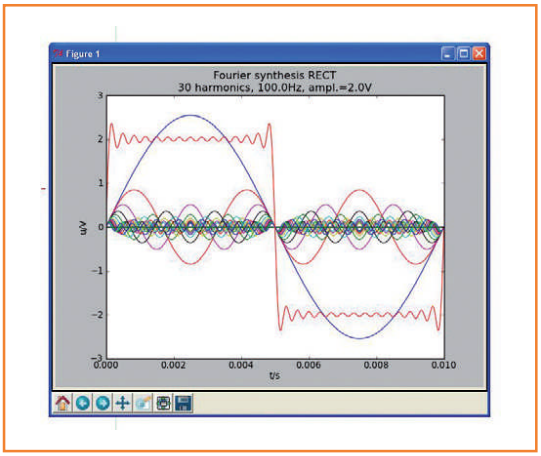


Figure 3.
Fourier synthesis of a square wave, see Listing 4.

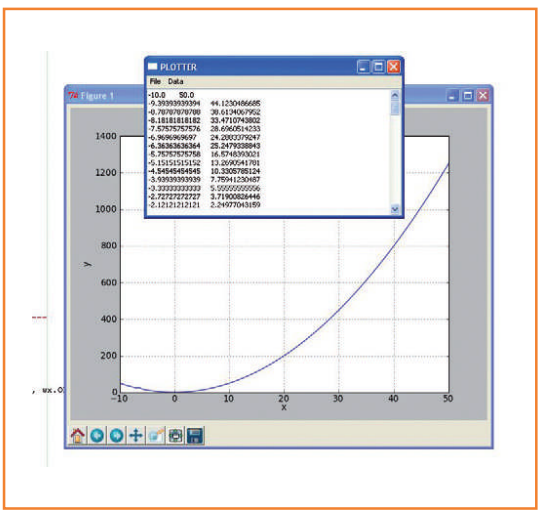


Figure 4.
Data plotter: Terminal window showing list of values and their XY plot.

make the plot by altering the number of harmonics defined in the code.

To GUI or not to GUI?

Dyed in the wool Linux users will tell you that any system with a GUI is not a serious operating system and that the computer mouse is in fact the work of the devil. Despite this view there are still many people who can appreciate easy to use, well designed software. Surely the graphical method of file selection is both convenient and these days almost indispensable?

Anyhow Python allows the combination of both graphical and non-graphical elements in the same program. The program in Listing 5 functions as a data plotter. Input data is displayed as both text in a terminal window using the Print command and also as a plot in a graphic window (Figure 4). Looking at the functions available in the GUI libraries you really are spoilt for choice. I began with the 'Tkinter' library because it was already supplied with the Python interpreter and proved easy to understand. However a problem occurred copying a graph to the clipboard so I moved on to use the 'wxPython' library in these examples. There are also other alternatives available such as 'PyQt' and 'GTK'.

This data plotter program is a useful example of the use of GUI elements in a program. It shows how measurement values stored in a text file can be edited and then displayed. The values

Listing 3: RC.py

```
import matplotlib.pyplot as plt
from numpy import pi, linspace, log10, logspace
from numpy import  complex, abs    # these allow
vector operations

#  EDIT
R = 10.0E3
C = 100.0E-9

# END EDIT

RC = R*C

# create f values equally spaced on a log scale
f = logspace ( 1, 4, 100)   # 100 values from 10**1
to 10**4

# calculate F (complex) and absolute value Fabs
F = 1 / (1 + 1j*  2 * pi * f * RC)
Fabs = abs(F)

# plot Fabs = f(f)
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(f, Fabs)
ax.grid(True, which = "both", linestyle = "-")
ax.set_xscale ("log")
ax.set_xlabel("f/Hz")
ax.set_ylabel("U2/U1")
ax.set_title("RC low pass frequency response")
plt.show()
```

are plotted on a two dimensional graph. The X and Y values are also displayed in the form of two columns.
In this example I have abandoned pure 'Pythonic' object orientated style for the sake of simplicity. After the necessary modules are imported all the functions are defined and 'main' is at the end of the program listing.
The main program begins with an instance 'app' of the App objects. Next a Frame object creates a window for the application. This window will have a menu and text box to enable editing and display of the data. The Text-Box is essentially a small editor: here you can edit data or enter new values, and the clipboard can be controlled using the standard key-sequence shortcuts. A right-click on the mouse pops up the classic editor functions.

Listing 4: Fourier.py

```
"""FOURIER SYNTHESIS OF A SQUARE WAVE"""
#-----------------------------------------------
-------
# EDIT HERE

n = 30                # number of harmonics
nb_points=1000           # horizontal resolution
frequency = 100.0   #Hz
amplitude = 2.0     #V

# END OF EDIT AREA
#-----------------------------------------------
--------
print "Importing modules"
import matplotlib.pyplot as plt
from numpy import sin, exp, linspace, pi
from numpy import zeros
#-----------------------------------------------
---------
def calc_amplitude(amplitude, i):
    """ Calculate amplitudes of harmonics"""
    # take only odd harmonics
    if i % 2 == 0:
        ai = 0
    else:
        ai = (4/pi)*amplitude / i
    return ai
#-----------------------------------------------
--------
def calc_harmonics(nb_points, n):
    """ Calculate harmonics and resulting voltage
    returns
    uharm = array nb_points * n
    ug = array nb_points
    """

    # init  arrays for resulting voltage and
harmonics
    ug = zeros(nb_points)
    uharm = zeros((nb_points, n+1))

    # harmonics  and total voltage
    for i in range(1,n+1):
        ai = calc_amplitude(amplitude, i)
        fi = frequency * i
        uharm[:,i] = ai * sin(2 * pi * fi * t )
        ug = ug + uharm[:,i]
    return uharm, ug
#-----------------------------------------------
----------
""" Main program"""

T=1/frequency

# equally spaced time array  for 1 period
t = linspace(0.0, T, nb_points)

# plot harmonics
uharm, ug = calc_harmonics(nb_points, n)
for i in range(1,n+1):
    plt.plot (t, uharm[:,i])

# plot resulting voltage
plt.plot (t, ug)

s=str(n)+" harmonics, "+str(frequency)+"Hz,
ampl.="+str(amplitude)+"V"
plt.title("Fourier synthesis RECT\n"+s)
plt.xlabel("t/s")
plt.ylabel("u/V")

# make plot visible
plt.show()
```

Listing 5: Dataplot.py

```
#!/usr/bin/env python
""" Plot data from file """
import wx
import os.path
import matplotlib.pyplot as plt

def create_menu(frame):
    # create menu
    menubar = wx.MenuBar()
    # main menus
    mnuFile = wx.Menu()
    mnuData = wx.Menu()
    menubar.Append(mnuFile,"&File")
    menubar.Append(mnuData,"&Data")
    # submenus
    m_Open = mnuFile.Append(-1,"&Open")
    mnuFile.AppendSeparator()
    m_Exit = mnuFile.Append(-1,"&Exit")
    m_Plot = mnuData.Append(-1,"&Plot")
    # attach menu to frame
    frame.SetMenuBar(menubar)

    # bind menu events to procedures
    frame.Bind(wx.EVT_MENU, OnExit, m_Exit)
    frame.Bind(wx.EVT_MENU, OnOpen, m_Open)
    frame.Bind(wx.EVT_MENU, OnPlot, m_Plot)
#---------------------------------------------------------------------
# Event handlers
def OnExit(event):
    frame.Close()

def OnOpen(event):
    # ask for filename
    dlg = wx.FileDialog(None,"Open data file", os.getcwd() ,"", "*.*", wx.OPEN)
    dlg.ShowModal()
    filename = dlg.GetPath()

    # open file, get data and  put it into textbox
    try:
        f = open(filename, "r")
        data = f.read()
        f.close()
        textbox.SetValue(data)
    except:
        wx.MessageBox("Could not open file!")
```

```
def OnPlot(event):
    # plot data
    x,y = fill_xy_with_values(textbox)
    plot_xy(x, y)
#---------------------------------------------------------------------
def fill_xy_with_values(textbox):
    # """ get values from textbox
    #    returns  arrays x, y  and number of data points"""
    text=textbox.GetValue()
    lines=text.splitlines()

    x=[]
    y=[]
    for line in lines:
        columns = line.split()      #separator can be one or more " " or "\t"
        x.append (float(columns[0]))
        y.append (float(columns[1]))

    return x,y
#---------------------------------------------------------------------
def plot_xy( x, y):
    """ Plot arrays x, y with matplotlib"""
    plt.figure(1)
    plt.subplot(111)                # 1 row, 1 col, plot nb. 1
    plt.grid(True)
    plt.plot(x, y)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.show()
#---------------------------------------------------------------------
#   Main
#---------------------------------------------------------------------
app = wx.App()

# create frame
frame = wx.Frame(None, title='PLOTTER', pos=(350,300))

create_menu(frame)
# editor textbox for data
textbox=wx.TextCtrl(frame, style = wx.TE_MULTILINE)

# show frame and run event loop
frame.Show()
app.MainLoop()
```

The 'frame.Show()' function displays a window and 'app.MainLoop()' is used to begin an event loop. The program waits for an event (mouse click or keyboard key press) and then branches to the event handler (the function called when an event occurs).
The first function defines the menu and binds menu events to the 'On' prefixed event handling procedures. Functions 'OnExit', 'OnOpen' and 'OnPlot' are called from the menu or when the corresponding key is pressed. 'OnOpen' calls the file selection dialog 'wx.FileDialog'. This also uses 'os.getcwd' (get current working directory) to find the file in the current directory.
When the file name is selected an attempt is made to open the file. A 'try – except' statement is used to detect any file opening error. When the file can be opened the contents are read into the variable 'data' and put into the text box.
Now the values can be edited, copied or pasted. Other values can of course also be added instead of storing them from a file.
The 'OnPlot' function displays the data as a graph. The functions 'fill_xy_with_values' and 'plot_xy' are also called here. The former reads data from the textbox and splits it into an array of lines. The 'for' loop is next iterated on all the lines which are split again using 'line.split()'. The individual values are then stored in the arrays 'x[]' and 'y[]'. These are then returned as the function values. The function 'plot_xy' draws the graph (Figure 4).
This small program is of course still a little rudimentary, but an equivalent in Visual BASIC would require much more work. It would be quite easy to add a menu option here to store edited data.

Coming up

By now you should have a basic appreciation of how powerful Python is and also how easy it can be to use. Python has a growing band of devotees, if you wish to explore there are many resources that will take you further than we have shown you in these examples. Python is well suited for tasks such as classic data collection and processing. Things like filters and FFTs are really no obstacle for Python and you have the possibility to upgrade your own project with a graphical user interface.
In the next installment we look into the programming intricacies of data acquisition and control via the RS485 Elektor Bus.

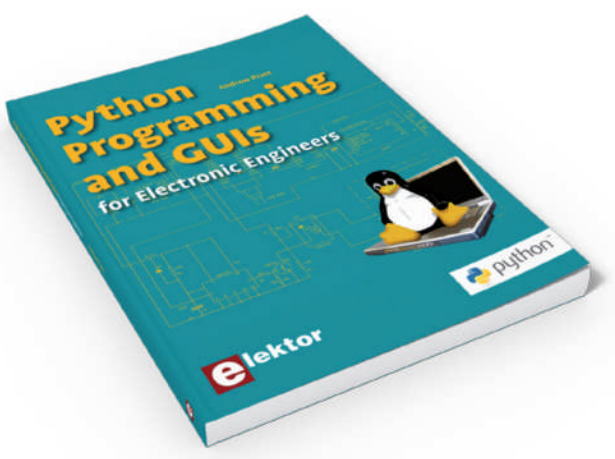
(120143)

The Author

Jean-Claude Feltes lectures in electronics at the Lycée Technique des Arts et Métiers in Luxembourg. This college caters for both arts and technical students and provides professional qualifications for apprentices and technicians. He spends much of his free time pursuing his interests in electronics and programming [2].

Internet Links & Literature

[1] Listings etc.:
www.elektor.com/120143
[2] The author's homepage:
http://staff.ltam.lu/feljc/home.html
[3] Python documentation:
https://pypi.python.org/pypi/RPi.GPIO
[4] Python tutorials:
www.awaretek.com/tutorials.html
[5] Introduction:
J.M. Hughes: „Real World Instrumentation with Python"
[6] Modules available in the Python package:
http://pypi.python.org/pypi
[7] Python for engineers:
Andrew Pratt: 'Python Programming and GUIs for Electronic Engineers'
www.elektor.com



ELEKTOR Preferred Suppliers

Tel. 1-978-281-7708 | **Fax 1-978-281-7706** | **Email ElektorUSA@smmarketing.us**

See what's brewing @ Elektor Labs 24/7

Check out
www.elektor-labs.com
and join, share, participate!

Beefing up DAC Resolution

More accuracy on your 8-bit conversions

By Olivier Schrevens
(France)

When you need resolution higher than 8 bits, the cost of mixed analogue/digital devices increases sharply. I wanted to show how, at no additional cost, I've improved the resolution of ordinary, cheap digital/analogue converters, by using the output of one as a programmable voltage reference for the others.

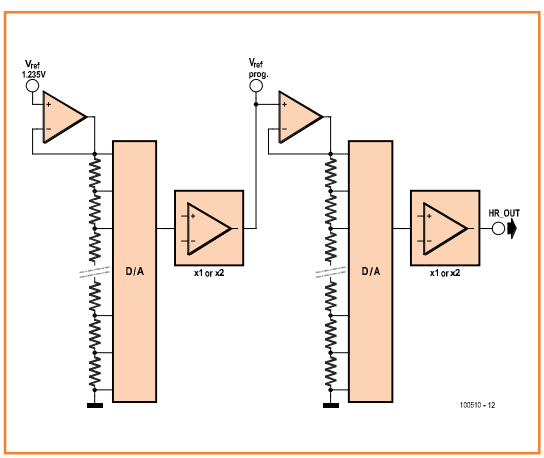


Figure 1. The principle of cascading D/A converters: the output of the first converter is used as the voltage reference for the second converter.

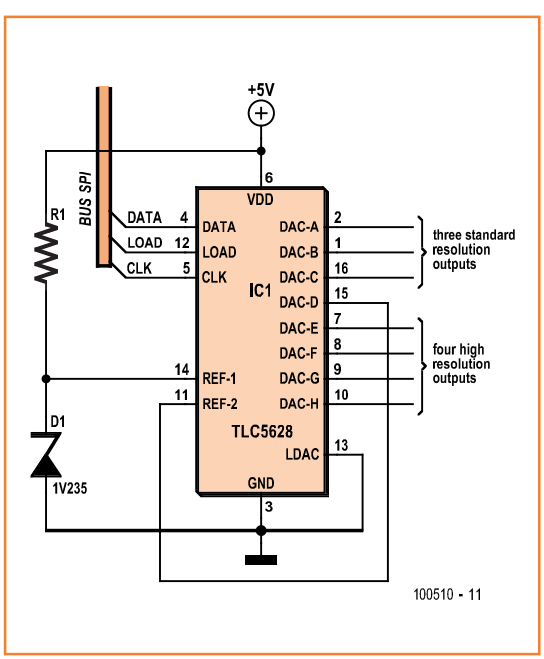


Figure 2. Configured in this way, the TLC5628 makes it possible to cover three voltage ranges, from 0 to 1.235 V, 0 to 2.47 V, and 0 to 4.94 V simultaneously, using 256 conversion steps.

The idea for this circuit came from the need to drive various types of moving-coil voltmeter, using a microcontroller, via digital/analogue converters with a series resistor on each output. Connected to an ATMega8, my initial circuit worked okay, but with the most sensitive voltmeters, the number of usable bits was not enough to achieve full scale deflection. Rather than use a specific output resistor for each voltmeter, I used the circuit described here to increase the resolution obtained using 8-bit converters. Instead of using voltage amplification, the trick consists in obtaining different output voltage ranges from the D/A converter by changing its reference voltage.

My main criterion in selecting the D/A converter concerns the minimum 1.5 mA current per output, to avoid the use of external opamps; as a secondary point, I felt that a single DIL package would make construction easier. I ended up choosing a TLC5628 from Texas Instruments, an 8-bit octal DAC in a DIP16 package, able to provide an output current of 2 mA per channel. Each converter channel includes a 256-step potential divider. The TLC5628 offers an option, crucial here, of doubling the gain by amplifying the converter output voltage (**Fig. 1**).

This device is controlled via a 3-wire serial programming bus (SPI), readily compatible with current microcontrollers. The command format consists of eight data bits, three bits for selecting each of the eight built-in converters, and lastly one bit for selecting the output voltage gain (×1 or ×2). Before

being validated all together by the LDAC signal, the SPI commands received by each of the converters in turn can be first stored in a latch. This option is not used here. You can find further details in the device datasheet.

As the circuit shows, the TLC5628 has two separate voltage reference inputs (pins 11 and 14), one for each group of four converters.

The external voltage reference common to the first four converters, here 1.235 V, provided by D1, an LM385, is connected to one end of the dividers, whose other ends are connected to ground. Hence each of the divider positions provides a voltage that is directly linked to the reference voltage.

Clearly, this is the feature we are going to be exploiting by using one of the D/A converters to programme the reference voltage for the other group of converters, thereby increasing the resolution.

The other three outputs from the first group can still be used with their normal 8-bit resolution. So ultimately, in combination with the output gain doubling option on each of the TLC5628's converters, the 8-bit resolution covers three voltage ranges from an initial 1.235 V reference. By combining the RNG (range) bits for the converter providing the reference voltage and for the channel being used, we obtain the following three ranges:

0 to 1.235 V
0 to 2.47 V
0 to 4.94 V

This is indeed equivalent to an increase in resolution. So in all we now have seven analogue outputs: four 'high resolution' outputs and three normal 8-bit outputs.

Of course, other types of converter may well be perfectly suitable for this circuit, if they have similar characteristics; for example, TLV5628, TLC5620, TLV5620, or TLV5604.
However, it should be noted that cascading two converters in this way has the disadvantage that their respective precision errors may add together.

What's more, by virtue of its architecture, this digital/analogue converter is more like a programmable potentiometer than a conventional DAC based on an R2R network.
As this trick works really well without the need to add any extra devices, I wanted to share it with Elektor readers. The test program [1] loops each of the three maximum output voltages onto output E (pin 7) for four seconds. The SPI clock is 11.0592 MHz/4. The SPI bus is in mode 3. An indication of the running of the program is provided via the serial port (38,400 baud, 8N1).

(100510)

Internet Link

[1] www.elektor.com/100510

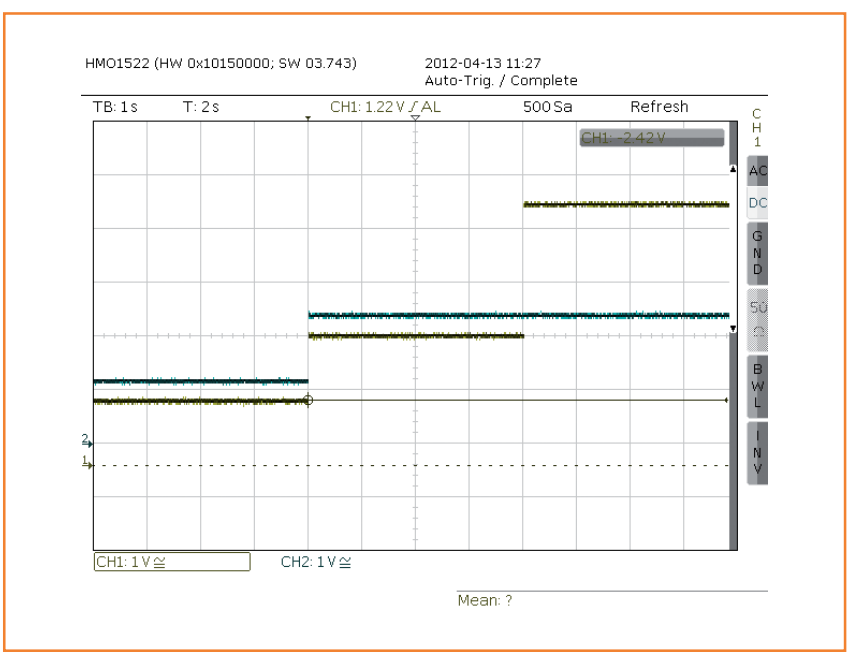


Figure 3. Result of the test program, displaying V_{ref2} and V_{out}.

On the left, V_{out} = 1.235 V and V_{ref2} = 1.235 V, Gain = 1
Centre, V_{out} = 2.47 V and V_{ref2} = 2.47 V, Gain = 1
On the right, V_{out} = 4.94 V and V_{ref2} = 2.47 V, Gain = 2

LCR Meter Shootout

By **Thijs Beckers**
(Elektor Editorial)

- "Wouldn't it be nice to compare our newest 500 ppm LCR Meter with one we have on loan from Hameg?"
- *"Yes, that could be interesting."*
- "I can bring a vintage meter from my Retronics stock, if you like."
- *"Awesome. Let's see how the three compare!"*

This recent *coffee chat* I had with a few colleagues at Elektor HQ resulted in me putting three LCR Meters with a completely different background through a quick hands-on test.

The Hameg meter (HM8118 [1], courtesy of Rohde & Schwarz Netherlands [2]) is a professional instrument with a price tag of $2,000+.

The vintage meter from the *Retronics* dept. is a General Radio Company (GRC) type 1650-A Impedance Bridge manufactured in 1960, costing a whopping $1,000 back then [3].

And finally, the Elektor 500 ppm LCR Meter [4], which should set you back less than $400 in parts.

I started off with the vintage GRC meter. Since the apparatus is older than I am and 'less than intuitive' in terms of operation — at least to me — I first needed to take a look at the manual. It took me quite some time to dial into the thing and get an accurate measurement readout. Here's a breakdown:

- read the manual: 10 mins;
- figure out how to measure a capacitor correctly: 5 mins;
- taking the actual measurement (setting the dials for 'zero' deflection): 3 mins.

To be fair, if you were to use the 1650-A on a daily basis, you would probably be able to shave off 80 % of the time needed for a single measurement. Still, that's over half a minute per measurement. I have to admit though, the looks, touch and feel of the sturdy dials and switches do add to the character of this piece of equipment.

Comparison chart

DUT	GRC 1650-A	Hameg HM8118	Elektor LCR Meter
Resistor 8.2 Ω	8.22 Ω	8.2458 Ω (V_x=31,86 mV, I_x=3.862 mA)	8.2379 Ω (V_m=31.13 mV, I_m=3.779 mA)
Capacitor 100 nF	100 nF *D*: 0.0267	102.03 nF *D*: 0.01205 R_s=19.20 Ω (V_x=364.9 mV, I_x=233.9 µA, 1 kHz)	102.20 nF *D*: 0.013 R_s=19,20 Ω (V_m=398.1 mV, I_x=256.0 µA, 1 kHz)
Capacitor 100 µF	92 µF *D*: 0.1	94.475 µF *D*: 0.10099 R_s=170.27 mΩ (V_x=7.301 mV, I_x=4.312 mA, 1 kHz)	92.400 µF *D*: 0.090 R_s=155.4 mΩ (V_m=7.076 mV, I_x=4.091 mA, 1 kHz)
Inductor 1 mH	1.255 mH *Q*: 0.63	995.85 µH *Q*: 0.49189 R_s=12.722 Ω (V_x=52.60 mV, I_x=3.711 mA, 1 kHz)	993.2 µH *Q*: 0.492 R_s=12.681 Ω (V_m=51.18 mV, I_x=3.621 mA, 1 kHz)

Then there's the HM8118. As a modern device, this one is (more or less) plug-and-play. After the auto calibration sequence the DUT (Device Under Test) is inserted into the test fixture (Hameg HZ181) and the requested component properties are instantly displayed on the blue backlit LCD. Estimated time: 40 seconds from start to finish, without reading the manual. Sequential measurements? Let's assume swapping the DUT takes three seconds, selecting the correct measurement unit another two, totaling five seconds.

Last but not least: the Elektor 500 ppm LCR Meter. With just an LCD, an on/off switch and five buttons this may be as far as plug-and-play can take you. The calibration is a once only job that had already been taken care of on the prototype I used for this test, so I could go straight to the actual measurement and put the DUT between the clips. Estimated time: 10 seconds.
Sequential measurements are probably a little more time demanding compared to the Hameg using the Kelvin clips, but the HZ181 test fixture can also be used on the Elektor LCR Meter.

Summarizing, ease of use has improved *a lot* since the 1960's and looking at the measurement results the Hameg and the Elektor LCR Meter kind of photo finished. Our little meter holds its own against the big gun (see the comparison chart). Of course it wouldn't be fair to ignore the extensive range of options on the Hameg and the 0.05% basic accuracy is nothing to sniff at either. Surprisingly, after more than 50 years and operated by a rookie, the GRC 1650-A isn't even that far off. Talk about a trusty lab companion. They don't make them like that anymore! Or do they?

Care to share your experience with 'vintage' lab devices? Drop us an email at editor@elektor.com.

(130166)

Internet Links

[1] www.hameg.com/13.0.html

[2] testenmeetwinkel.nl

[3] www.elektor.com/075064

[4] www.elektor.com/110758

New Performance Requirements for Resistors

By
Dominique Vignolo
(Vishay)

in aeronautics applications

The designs of today's aircrafts are being driven by two primary goals: increasing aircraft fuel efficiency and conforming to anti-pollution regulations. The humble resistor can be helpful, provided some of its key specifications are given close consideration.

In order to increase fuel efficiency, the weight of the aircraft is decreased by reducing cabling, which can be achieved by moving electronics close to their function. To comply with anti-pollution regulations, electric engines are being used to move the aircraft on the ground.
In addition to changing design, the new demands placed on aircraft manufacturers have also created new performance requirements for electronic components, including resistors. In this article, we will explore the required parameters for different types of resistors, including high-temperature capabilities for stringent operating conditions and long-term stability.

A history of high-temperature components in aeronautics applications

Over the past eight years, aircraft manufacturers have used high-temperature parts in a number of applications. One of these was landing and braking monitoring systems, where brake temperatures were measured, and Wheatstone bridges were used to monitor hydraulic and tire pressure. In this type of application, the electronics were located in the wheel, and high temperatures reached them within an hour.
In terms of performance, these systems required components with operating temperature ranges from –55 °C to +175 °C, but this quickly needed to be expanded to +200 °C. Components with good long-term stability were also required, as the measurements had to remain stable for the life of the aircraft. The expected drift after several thousand hours of life could not exceed a given percent. Finally, the components had to exhibit good behavior during acceleration, vibration, and

harsh environments. SMD products were shown to be the best under such conditions.
Like the aircraft braking monitoring systems, this sensor required components with an operating temperature range from –55 °C to +200 °C, very good long-term stability, and excellent behavior during acceleration, vibration, and harsh environments. The application utilized SMD wraparound chip resistors.

New regulations

With new regulations aimed at reducing pollution and saving fuel, more and more high- temperature applications are showing up. For example, engine temperatures are monitored so they can be regulated by a computer. This means electronics can be found inside the engine, where temperature can be very high. Taking into account that the average life of an aircraft is 25 to 30 years, the load-life stability of the components used at high temperatures is a key parameter for aeronautics applications. The goal is to find the best compromise between handling the power and enhancing long-term stability.
Likewise, sensors can be used to measure temperature of helicopter turbines.

Thermal management

Referring to **Figure 1**, resistor manufacturers need only take care of $R_{th(jsp)}$, but must carefully consider their choice of material, the resistor pattern, terminations, etc. Manufacturers who also improve thermal stability can offer resistors that can withstand higher and higher temperatures without undergoing significant drifts. This removes limitations on T_j.
The control of all the others parameters — namely T_a, P_d, and $R_{th(spa)}$ — are addressed by the customer's assembly designers. Designers must take the PCB material, the thickness and layout of the copper tracks, the cooling system, and the interaction between surrounding components into consideration.

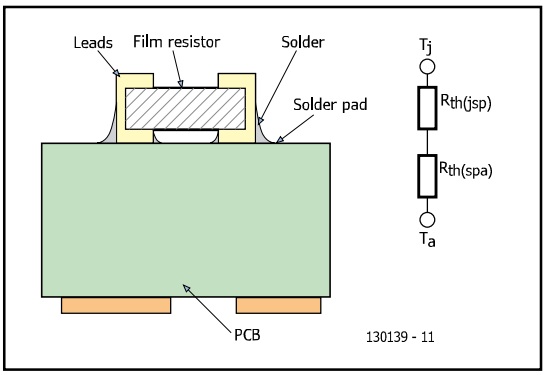


Figure 1.
Thermal parameters for a wraparound chip resistor.

A poor thermal management might induce melting or reduced reliability of the solder joints; reduce PCB performance (even burn-out); and lower chip resistor performance.

$R_{th(jsp)}$ and experimental data

To use the thermal model above, manufacturers need to provide $R_{th(jsp)}$ for standard termination parts, in addition to experimental data relevant to chip resistors of standard sizes mounted on various PCBs. These PCB should be chosen to represent the standard and best cases in terms of thermal resistance.
In the experimental data collected in **Table 1**, we have:

- PCB sCu — A PCB with a thickness of 1.6 mm, double sided, 35 µm thick copper (minimum), at least 50 % copper coverage both sides
- PCB MCu — A PCB with a thickness of 1.6 mm, double sided, 70 µm thick copper (minimum), at least 80 % copper coverage both sides
- Temperature versus drift was plotted against time and appears in **Figure 1**.

Derating curve of a basic thermal model

The derating curve in **Figure 2** is a representation of a basic thermal model:

Table 1. Load-Life drifts after 15,000 hours at various temperatures (experimental data).

Size	$R_{th(jsp)}$ (°C/W)	PCB sCu	PCB Mcu
		$R_{th(ja)}$ (°C/W)	$R_{th(ja)}$ (°C/W)
0603	27	200	67
1206	20	110	60
2010	12	95	52
2512	11	95	51

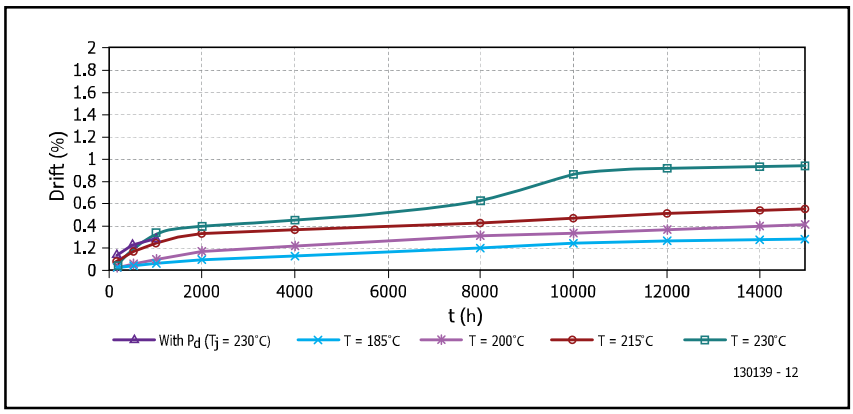


Figure 2.
High-temperature drift vs. time.

$T_c = T_a + R_{th} \times P_d$

where
T_c = temperature to be controlled;
T_a = ambient temperature;
P_d = maximum allowed power dissipation;
R_{th} = thermal resistance between the surface of the resistor at temperature T_c, and the ambient.

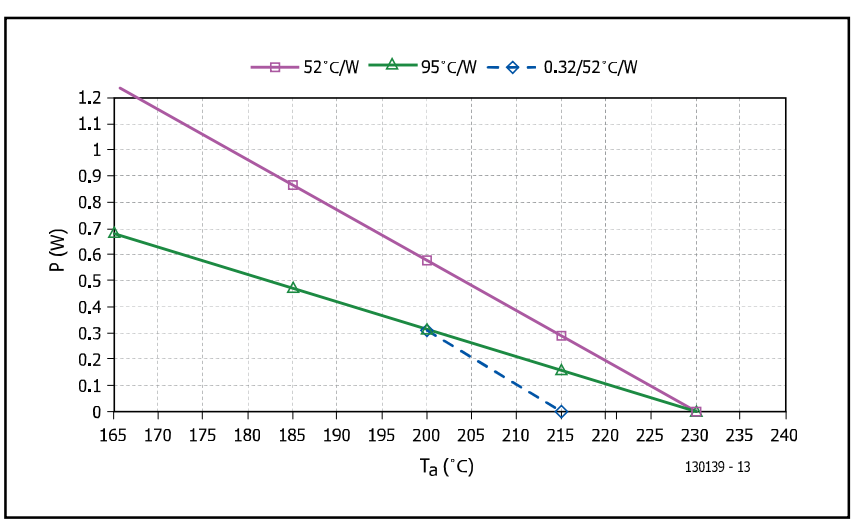
The model can be written as $P_d = (T_c - T_a) / R_{th}$. Per Table 1 we get: R_{thja} = 52 °C/W for a P2010 chip on a MCu PCB, and R_{thja} = 95 °C/W for a P2010 chip on an sCu PCB.

Using the derating curve

Providing $T_{j\,max}$ = +230 °C, the maximum power dissipation of the resistor at T_a = +200 °C will be:

- 0.57 W for R_{thp} = 52 °C/W — This is the *best assembly*.
- 0.32 W for R_{thp} = 95 °C/W — This is the *standard assembly*.

Figure 3.
Example of a derating curve (P2010).



The first way to use the derating curve is to check the maximum power rating that can be applied at a given temperature. For instance, if a customer uses the *best assembly* (52 °C/W), the maximum power at +200 °C will be 0.57 W. The second way is to reduce drifts by limiting the temperature at the surface of the resistor. In this example, the best assembly is used, but the customer limits the power to 0.32 W. This moves the 52 °C/W curve down and the junction temperature will be +215 °C instead of +230 °C as with the 52 °C/W curve.

Conclusion

From an analysis of temperature-induced drifts, we have pointed out some specific features of our thin film resistors that give them advantages for high-temperature applications.

The irreversible drifts other than load life are negligible. The load-life drift depends on T_j, however it is reached; by pure ambient temperature or the sum of ambient temperature and power dissipation ($T_j = T_a + R_{thja} \times P_d$). This is valid, providing some P_d limitations given in the datasheets. From an analysis of actual stability data and drifts versus time for various temperatures, it is obvious that even for T_j as high as +230 °C, drifts are under control and rather predictable from manufacturing data processes.

To help assembly designers we developed a thermal model showing thermal resistance figures necessary to use this model. The derating curves illustrate how good thermal management leads to load-life drift minimization.

From the above derating curves, it is clear that the load-life stability of the resistor or of the resistor network is enhanced by properly controlling the temperature at the surface of the resistor, thus increasing the life of the components in extreme operating conditions. Such conditions are becoming more common as electronics in aeronautics applications move closer to their functions.

(130139)

iFixit (!) admit defeat to Orange repairability

The guys at iFixit have been accused of favoring one fruit above all others. Now, to show some appreciation towards other members of the fruit family, they decided to test the accessibility, repairability, and end-of-life design of the Orange.

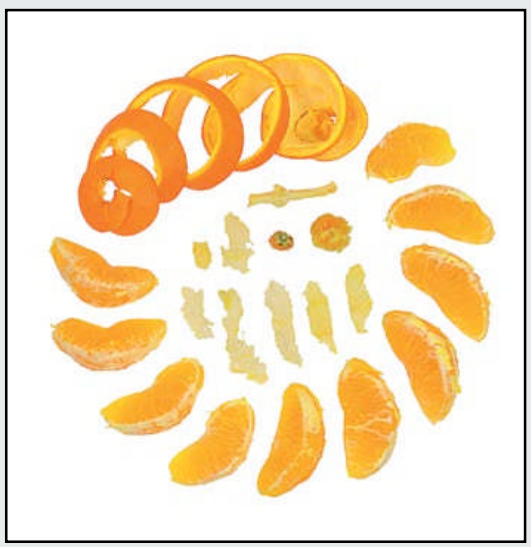
The Orange is a fickle fruit. Believed to be the hybrid between a pomelo and a mandarin, the team at iFixit had no luck opening it with a guitar pick, plastic opening tool, tech knife, ruler, pair of tweezers, or even an iSesamo! After almost losing all hope, a tool descended upon them from the repair heavens, with Morgan Freeman's voice booming throughout our teardown room. They called it the oOpener!

The Orange received a first-ever 0 out of 10 Repairability Score for a variety of reasons, but mainly because it's... an orange. Lest you think that IFixit is biased against oranges, note that users will have to break its outer case in order to open it, the device is impossible to reassemble, and the internal components are filled with acid. Sounds like a bad time all around.

Some people may have tried the teardown at home with mild to moderate success. They can celebrate their success with an Orange Teardown t-shirt! Or if they're having technical difficulties with their Orange, they can make sure they've identified their Orange correctly, and then visit iFixit's Orange Troubleshooting Guide to figure out what's going on. Or perhaps they just need the right tool for the job.

Having no luck with our vast array of poking, prodding, and prying tools, a little more heat might be necessary... But it turns out keeping a heat gun on the same spot for too long will damage the outer layer of the peel beyond repair. If you're not extremely cautious, you will find yourself in need of a new outer peel.

As an added bonus, the Orange comes with a free accessory, the Orange Peel — an amusing, Slinky-like contraption, guaranteed to deliver hours of fun.



All ten major internal components of this device are easily removable. However, iFixit are worried that they might not go back together as easily as they come apart, a common problem in fruit repair.

Though the Orange's repairability is highly questionable, iFixit do admire its end-of-life design. It is completely recyclable, compostable, and delicious-able. We hope that devices like this will someday catch the attention of other device manufacturers and help keep electronics out of landfills... unless they are compostable, of course!

Navel Oranges have small, sterile seeds that cannot be used to reproduce more Oranges — a cutting from an existing fruit-bearing tree must be grafted to another tree. This is the most advanced anti-piracy DRM measure iFixit have seen, definitively ensuring that Orange owners won't be able to produce copies of their Orange to share with their friends.

www.ifixit.com/Teardown/Orange+Teardown/13470/1

(130028-IV)

Model car sector gets processing power and software flexibility

Nordic Semiconductor ASA announced that Taiwanese original design manufacturer (ODM), Relight Technologies, has selected the nRF51822 Bluetooth® low energy and 2.4 GHz proprietary System-on-Chip (SoC) and the nRF24LE1 SoC as the basis of a high-end remote control model car radio transmitter. Relight's "Smart-Tx" is one of the most advanced surface radio transmitters for remote control model car drivers.

The nRF24LE1, a proven 2.4 GHz ULP SoC used in millions of devices worldwide powers the wireless link between the transmitter and vehicle. Relight selected the chip because of its robust RF performance and because it can be programmed with the ODM's own RF software protocol that has been specifically designed to cope with the demands of remote control cars used by enthusiasts. These vehicles can reach speeds approaching 60 mph and are often surrounded by many other remote control vehicles, so range, low latency, and interference immunity are essential requirements for the wireless link.

The remote control handset also employs Nordic's nRF51822 SoC. Apart from its superior RF performance and ultra-low power consumption (–92.5dB RX sensitivity in Bluetooth low energy mode and sub-10mA peak currents when running off a 3 V coin cell battery), Relight selected the chip for two key reasons: Seamless connectivity with smartphones, and because the device's powerful 32-bit ARM® Cortex™-M0 based processor features ample computing overhead to cope with the complex remote control algorithms.

In addition to providing the remote controller's computing capability, the nRF51822 powers the wireless link between controller and smartphone that relays the user's set-up instructions after they have been selected using the app. The iPhone is a Bluetooth Smart Ready product, ensuring it is interoperable with any Bluetooth Smart product. A Bluetooth low energy radio — as integrated into the nRF51822 — is a prerequisite for a Bluetooth Smart product.

www.nordicsemi.com

(130028-XI)

New industrial analog I/O server

Lantronix'® newest member of its xSenso™ analog sensor networking family is designed specifically for use in rugged and harsh environments including industrial automation, process control, manufacturing, chemicals, oil and gas industries, and many more. The new analog and relay outputs provide the ability to take action by instantly controlling industrial processes and equipment based on the sensor readings and predefined thresholds to solve real-time problems. The Lantronix xSenso product family comprises feature-rich, low cost solutions for remote sensor monitoring and process control. The new xSenso Controller is a compact, DIN-rail or wall mount solution that enables analog sensors (voltage or current) to easily and transparently send real-time data to any node on the Internet or to a cloud-based application. In addition to its already robust feature set, the new xSenso can take action when a condition is met (e.g., temperature threshold, pressure level, etc.), triggering its internal relays or outputting voltage or current to control an event in real time. In an existing control system, the xSenso can also be placed in between the sensor and the controller to mirror the analog data from its input to its output, while simultaneously presenting the data through a web browser to computers or mobile devices.

The product family is compatible with some of the most popular off-the-shelf data acquisition systems such as LabVIEW™ and DASYLab™, and its compact and efficient design enables it to be affordably installed in dispersed or remote locations.

www.lantronix.com (130028-VI)

RFICs flying in Globalstar communication satellites

Peregrine Semiconductor UltraCMOS® Phase Locked Loop (PLL) frequency synthesizer and prescaler devices are designed into six Globalstar mobile communication satellites that were launched into orbit on February 6, 2013. Built by Thales Alenia Space in France, the low-Earth orbit satellites transmit audio and data communications for Globalstar's mobile voice and data customers worldwide. Peregrine's PLL and prescaler enable communication in sixteen C- and S-band transponders in the system, which connects end users with terrestrial communication networks via vehicle-mounted mobile devices, as well as fixed terminals, such as those used for rural telephony. The Peregrine devices feature extremely low phase noise and Single Event Effect (SEE) immunity — attributes enabled by the insulating properties of the UltraCMOS process.

Single Event Effects are errors that are caused by naturally-occurring space-based radiation. There are two primary types of SEEs. Single Event Upsets (SEUs) are non-destructive and can be corrected. Single Event Latchups (SELs), on the other hand, are often catastrophic, resulting in permanent damage and requiring, at a minimum, a power-down to recover. SELs can occur when a high-energy particle strikes a semiconductor device, causing a short circuit from power to ground within the device. RFICs manufactured using UltraCMOS technology do not contain the bulk parasitics found in regular CMOS devices, making latchup impossible.

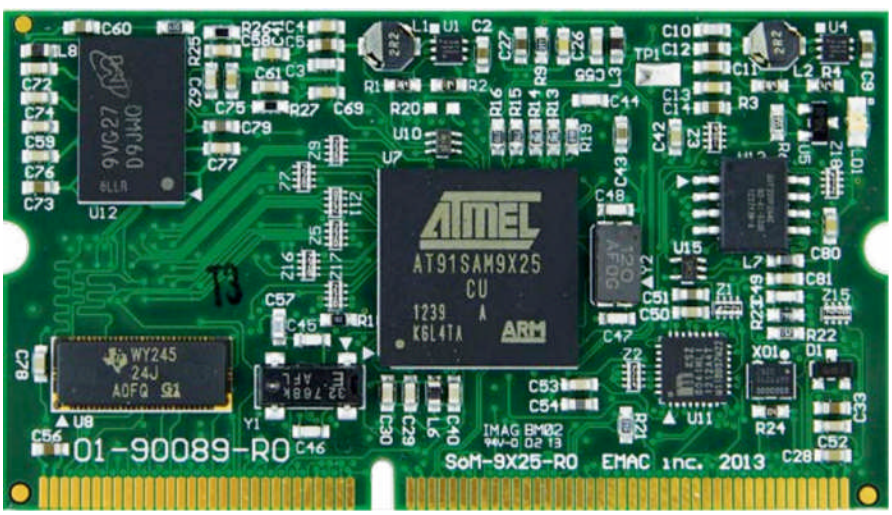
Peregrine's UltraCMOS technology is an advanced RF Silicon-On-Insulator process that utilizes a synthetic sapphire substrate — a near-perfect electrical insulator. This substrate enables low parasitic capacitance, high signal isolation, excellent broadband linearity, and inherent SEL immunity. These attributes make UltraCMOS well suited for high-reliability applications, such as commercial satellites.

www.psemi.com (130028-X)

Wide-temperature ARM system on module

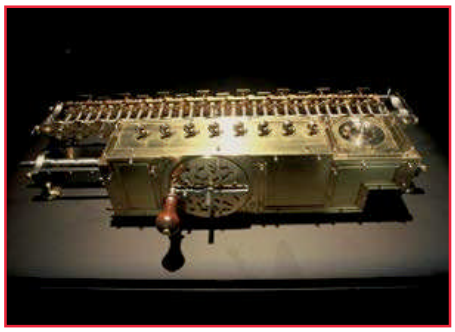
EMAC, Inc. introduces the SoM-9X25, a wide temperature System on Module (SoM) based on the Atmel AT91SAM9X25 processor. Designed and manufactured in the USA; this wide temperature, fanless ARM9 400 MHz SoM has an Ethernet PHY included along with 6 serial ports with auto RS-485 provision. It utilizes up to 1GB of NAND Flash, 8MB of serial data flash, and up to 128MB of DDR2 RAM. A SoM is a small embedded module that contains the core of a microprocessor system.

Using the same small 144 pin SODIMM form-factor (2.66" x 1.5") utilized by other EMAC SoM modules, the SoM-9X25 is the ideal processor engine for your next design. All of the ARM processor core is included on this tiny board including: flash, memory, serial ports, Ethernet, SPI, I²C, I²S audio, CAN 2.0B SDIO, PWMs, timer/counters, A/D, digital I/O lines, video, clock/calendar, and more.
The SoM-9X25 is designed to plug into a carrier board that contains all the connectors and any custom I/O required for the application. This approach allows the customer or EMAC to design a custom carrier board that meets the customer's I/O, dimensional, and connector requirements without having to worry about the processor, memory, and standard I/O



functionality. With this System on Module approach, a semi-custom hardware platform can be developed in as little as a month.
In addition to the option of the developing a Carrier board, one can be purchased off-the-shelf from EMAC. EMAC provides off-the-shelf Carrier boards that feature A/D, D/A, MMC/SD card, keypad, LCD, audio, and modem interfaces. The recommended off-the-shelf carrier board for the SoM-9X25 is the SoM-150ES which allows the user to immediately start coding their application using the powerful Linux or WinCE operating system and tools. The System on Module approach provides the flexibility of a fully customized product at a greatly reduced cost.
EMAC provides a Free Eclipse IDE for Linux development. All the compiling, linking, downloading, and debugging inherent to software development can be done from one easy to use high level interface. When developing for Microsoft Windows CE 6.0 applications, Microsoft Visual Studio 2005/2008 can be utilized. Both Visual Studio and Eclipse provide everything the user needs for developing SoM-9X25 applications. EMAC provides an SDK for the SoM-9X25, which contains source examples and drivers. Quantity 1 price for SoM-9x25 starts at $180.

www.emacinc.com/som/som9x25.htm
(130028-IX)



1

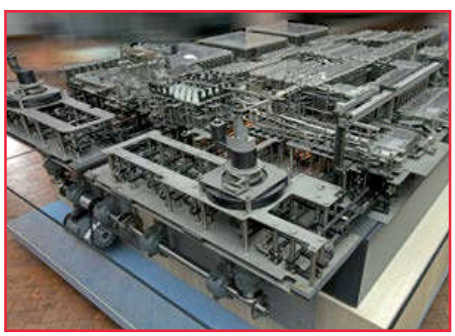
Konrad Zuse's

Z1 through Z4, and Beyond

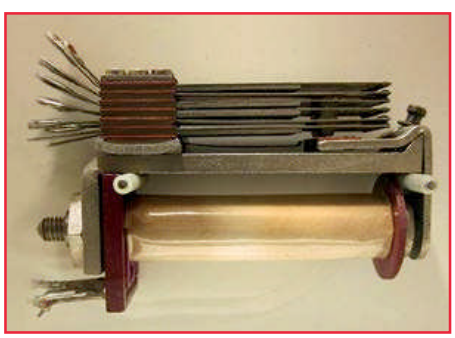
It all began with zeroes,ones and algebra

2

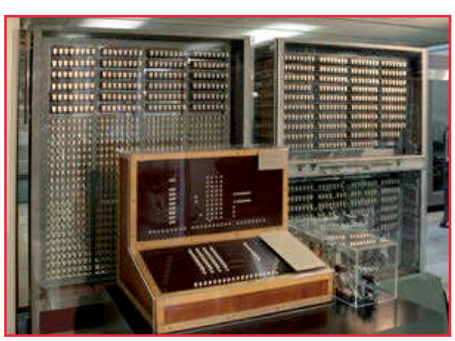
Ever since the earliest days of mathematics and logical thought, people have tried to find ways to simplify the repetitive work involved. Based on an understanding of logical functions and relationships various highly sophisticated calculating machines were developed. This article takes a tour of Konrad Zuse's impressive contributions to the development of the computer.



3



4



5

By **Peter Beil** (Germany)

One of the earliest calculating machines is the one built by Gottfried Leibniz in around 1700 (**Figure 1**). However, a couple of hundred years passed before calculating machines were able to carry out logical operations directly. The resourceful thinker and inventor Konrad Zuse (**Figure 2**) built his Z1 in 1938. This was a fully-fledged (albeit mechanical) computer. (The term 'computer' derives from the Latin for 'reckon together' or 'calculate'.) The machine was freely programmable and calculated using binary values. The original was destroyed in the war, but a replica has been made and is on show in the Deutsches Technikmuseum (German Technology Museum) in Berlin (**Figure 3**).

Z1: a troubled start

The Z1 computer was based on logical functions such as 'and' and 'or', these functions being computed purely mechanically! The machine had a considerable number of infelicities arising as a result of mechanical tolerances, friction and so on, and the switching elements often jammed. One advantage the Z1 had over more recent computers is that its storage was completely non-volatile: modern machines can only emulate such a feature indirectly.

Konrad Zuse had seen that the only way to make the required operations technically feasible in his computer design was to use the binary system. This was in sharp contrast to the researchers and pioneers in the USA and in Great Britain, who initially preferred to use the decimal system.

Z2, 16 bit @ 10 Hz

The mechanical problems with the Z1 prompted Zuse to try an experiment: in 1939 he designed the Z2, employing several hundred telephone relays (**Figure 4**). It ran at a clock frequency of approximately 10 Hz, and provided the four basic arithmetic operations on fixed-point binary numbers. It sported sixteen-bit memory and weighed around 300 kg (660 lb).

Z3: data from filmstrip

In May 1941 Zuse demonstrated his Z3, the first viable digital computer (**Figure 5**). This used 600 relays in the arithmetic unit and a further 1400 relays in the memory unit. Like the Z1, it used binary floating-point arithmetic, and it was the first universal programmable computer.

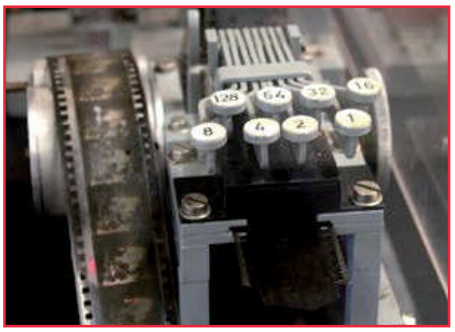
Reportedly Zuse had a friend working at UFA (a large German film studio) who gave him the idea of using punched filmstrip as an input medium.
Punching the filmstrip was done directly on the unit (**Figure 6**). There was also of course an accompanying reader: this used the sprocket holes in the film to obtain precise alignment with the punched holes representing the input data (**Figure 7**).
The original of the Z3 was also destroyed in the war, but a fully-functional replica is on display at the Deutsches Museum (German Museum of Science and Technology) in Munich. The replica was made by (now dissolved) company Zuse KG. A glance at its innards reveals a phenomenal amount of wiring, a rotary switching mechanism to generate clock signals, and stepping relays, as subsequently used in telephone dialing systems (**Figures 8**, **9** and **10**). The machine operated on 22-bit words, comprising a seven-bit exponent, a 14-bit mantissa and a sign bit. It had relay-based storage for 64 words, and programs always ran in a loop. Numbers could be entered using a keyboard (**Figure 11**), and results appeared in a display area using small lamps (**Figure 12**). As well as the four basic arithmetic operations, the Z3 also offered a square root function.

Z4: still all mechanical

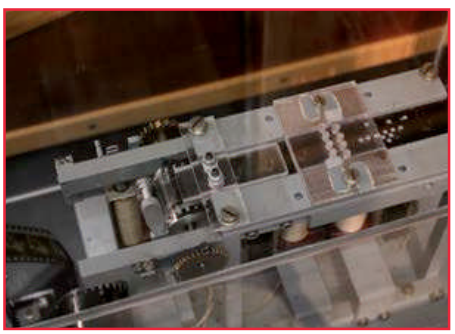
In 1942 Zuse began work on the Z4, which was first ready for use in 1945 (**Figure 13**). This machine was also based on relays and found itself in competition with the American 'Mark I' (1944) and 'ENIAC' (1946). However, these operated on different principles, for example using valve technology and decimal representations (**Figure 14,** Einiac 1946).
Like the Z3, the Z4 was very similar in many ways to today's computers. It could execute loops, offered floating-point arithmetic with a 24-bit mantissa, a 7-bit exponent and a sign bit (and hence a basic 32-bit word length), and two simultaneously-operating arithmetic units. As well as the basic arithmetic operations it could calculate squares and extract roots. The Z4's instruction set consisted of a total of 29 instructions. An electric printing device was connected to it, but it was not what we would now recognize as a 'printer' but rather was used to produce logs of the machine's computations (**Figure 15**).
Programs were again entered using punched filmstrips (**Figure 16**). 'Storage' in the modern sense was not possible, and interrupting the power supply resulted in the loss of all data. Ferrite core memory was not to appear on the scene until the middle of the 1950s.
The Z4 was capable of about 30 operations per minute: an addition took half a second and a multiplication about 3.5 seconds. Work had started on various refinements, such as program branches and index registers for address calculations, but these were lost in the chaos of war.

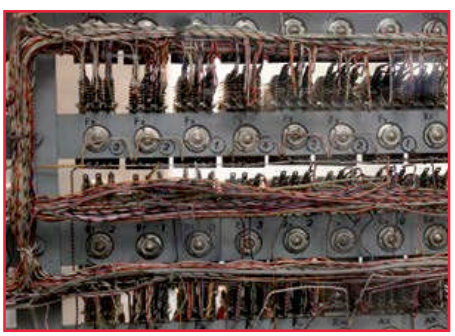
Programming language and post-war developments

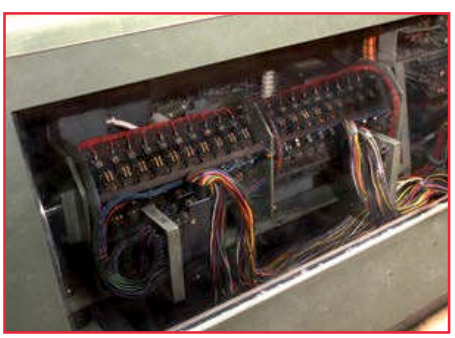
It is not widely known that Konrad Zuse was already at that time aware of the need for a high-level programming language. In 1945 and 1946 he developed 'Plankalkül' ('formal system for planning') but he was not able to publish his results. The work foreshadowed more modern programming languages such as Fortran, ALGOL and COBOL.
In the years immediately following the war the Z4 went on something of an odyssey through Germany, Switzerland and France. IBM (among others) was interested in acquiring the intellectual property rights with the aim of hampering further development of the machine. By the 1950s the machine found itself at the ETH (Swiss Federal Institute of Technology) in Zurich where it was employed to help with scientific research problems: in 1950 it was the only computer operating in continental Europe. In 1960 the machine completed its travels, coming to rest in the Deutsches Museum in Munich. The Zuse company continued to make scientific mainframe computers for several years, and in 1961 produced the

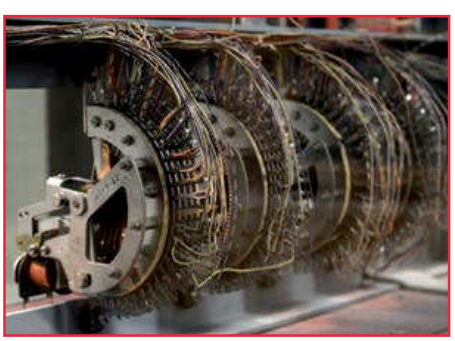


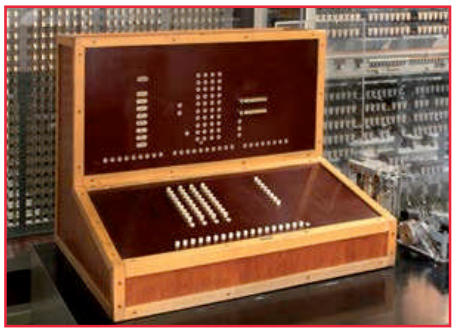
6


7

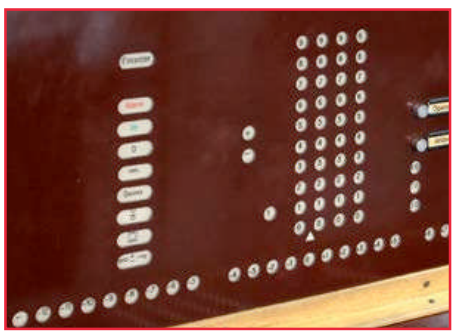

8


9


10



11

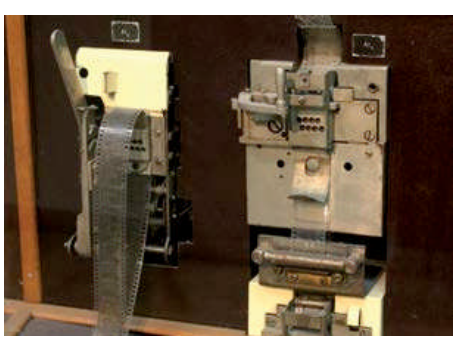


12

13

14

15



16

How a computer like the Z3 'thinks'

K. Zuse (l) and H. Nixdorf

Each operation ultimately comes down to the addition or subtraction or two integers. Two floating-point numbers are added as follows. First the difference between the exponents is calculated; then this value is used to shift one of the mantissas to align the binary points; and then the aligned mantissas are added. Subtraction is performed similarly, with an additional step where the two's complement of the second mantissa is taken.
Multiplication is done by adding the two exponents and then multiplying the mantissas using an iterative addition method.
Division is similar to multiplication: the exponents are subtracted and the mantissas then divided using an iterative subtraction method.
Square roots are calculated using an iterative method similar to division.
At a high level the arithmetic unit consists of two parts, one dealing with calculations on the exponents and one with calculations on the mantissas.
Instructions that are implemented using an iterative algorithm require a sequencer to drive the separate parts of the machine: this roughly corresponds to the use of microcode in modern processors.

first ever fully-functional plotter in the form of the 'Graphomat'. However, the small company was unable to withstand the overwhelming competition from the United States and in 1964 was taken over by BBC (Brown Boveri & Cie.) and then in 1967 by Siemens AG.

(130040)

Acknowledgements

Photographs 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16 by the author, with the permission of the Deutsches Museum in Munich:

Photograph 14: wikipedia.de

Photograph 3: German Technology Museum, Berlin

Photograph 2: Prof. Horst Zuse

ESTD 2004

Retronics is a monthly section covering vintage electronics including legendary Elektor designs. Contributions, suggestions and requests are welcome; please telegraph editor@elektor.com

Hexadoku Puzzle with an electronic touch

This Hexadoku puzzle doesn't require anything remotely resembling a Zuse "Retronics" computer to crack. Logic reasoning, concentration and patience should do the trick and produce the solution. So go ahead entering the right numbers or letters A-F in the open boxes, find the solution in the gray boxes, submit online, and you automatically enter the prize draw for one of four 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 one Eurocircuits PCB voucher worth **$140.00** and three Elektor book vouchers worth **$60.00** each, which should encourage all Elektor readers to participate.

Participate!

Before July 1, 2013, supply your personal details and the solution (the numbers in the gray boxes) to the web form at

www.elektor.com/hexadoku

Prize winners

The solution of the April 2013 Hexadoku is: **934CB.** The Eurocircuits $140.00 voucher has been awarded to David Smart (USA). The Elektor $60.00 book vouchers have been awarded to Joseph Reding (Luxembourg), Karsten Krummeich (Germany), and Paul Blaak (Netherlands).

Congratulations everyone!

5		D		9	0		A	6	7		B		1		
C	E		3		8		5						A		
		2								1		F		0	9
A				7		C	F			D			4		
F	4	C		3		5	8		E						1
0			6		7		B	C	D	9		2	5		
E	7		2				1	4		8	A				D
	1		B						F	5					8
		3	7	5			D		1	2	8	0		A	B
			5	B		7					D	C			
2	6	1			C					B				D	3
B		E				8	3				9	1			7
			F			B	6	9	8	E				1	
	8			1	5		E				4		B		0
		9			2			D	B		6			3	
				8	D				5	F	7	9		4	A

9	C	2	4	B	D	E	F	7	6	A	0	5	8	3	1
5	E	B	D	3	1	A	9	4	F	8	C	6	7	0	2
A	F	3	0	8	4	6	7	D	1	2	5	E	9	B	C
6	1	7	8	C	0	5	2	E	3	9	B	4	A	D	F
7	B	6	E	4	A	F	1	9	8	0	3	C	D	2	5
8	D	9	3	5	2	B	0	6	A	C	1	7	E	F	4
C	0	1	F	6	8	7	E	5	4	D	2	A	B	9	3
2	A	4	5	9	C	D	3	F	7	B	E	8	0	1	6
3	2	8	1	A	5	0	D	B	C	E	F	9	6	4	7
B	4	5	7	E	F	8	6	0	2	3	9	1	C	A	D
D	6	F	A	2	7	9	C	1	5	4	8	B	3	E	0
E	9	0	C	1	3	4	B	A	D	6	7	F	2	5	8
F	3	E	B	7	6	C	A	2	0	5	4	D	1	8	9
0	5	C	6	D	E	1	8	3	9	F	A	2	4	7	B
1	8	A	2	F	9	3	4	C	B	7	D	0	5	6	E
4	7	D	9	0	B	2	5	8	E	1	6	3	F	C	A

The competition is not open to employees of Elektor International Media, its business partners and/or associated publishing houses.

In The Clouds

By **Gerard Fonte** (USA)

The Leading Edge

Everyone likes to be at the forefront of technology. Using all the latest gadgets and applying all the latest ideas. Unless you're a company and you have to rip up and throw away all the old technology for the newest. That's expensive. And will it really give you the edge? Or is this just a fad that you'll regret in a year or so?

Re-training the engineers and technicians and assemblers is not cheap. And then there are all those technical issues. Your engineers haven't designed this type of system before. What problems will arise? Worse, what problems will go unseen until the product has been marketed for a year or so? It's not called the "Bleeding Edge" for nothing. This is where the company hemorrhages capital and time for a perceived advantage somewhere in the future. If it works, it's brilliant. If it doesn't, pass out the parachutes.

In the 1950's atomic power was The Thing. Ford developed the 1958 concept car called the Nucleon that was to operate on a small atomic fission reactor (similar to those used on submarines). Can you imagine the results of millions of unlicensed nuclear reactors tearing across the US, getting into accidents and spilling their nuclear fuel? The idea seems mind-boggling today. In 1954 RCA considered an atomic battery for use with hearing aids. What could possibly go wrong with strapping a radioactive device to your head for 12 hours a day—for the rest of your life?

Today's Thing is cloud computing. There's all this "excess hardware and software" that's just sitting around and not being used. Why not take advantage of all these cheap resources? You don't have to expand your own facilities. All you have to do is rent them cheaply on the internet. No re-training costs, no hardware costs, no added personnel. You get instant leading-edge technology with no risk and fantastic benefits. It seems like a no-brainer decision. (For reference, atomic power was billed as being "too cheap to meter." That's right—free energy.)

The Fine Print

Except you no longer have control of your data and software. Someone else does. If you are just playing a game on-line, control probably isn't very important. The worst that could happen is that your avatar could face an untimely and unexpected termination. But what about your proprietary design information or financial records? You are now dependent on another party to keep your data safe and sound.

Of course the companies that supply cloud-computing resources take a great deal of effort to maintain the highest security for their data. They usually make at least a second copy of your data in case of a hardware failure. They have major assets in place to prevent unauthorized access and infection from viruses and malware. After all, this is their business. And if they can't maintain security, they know that they will be quickly out of business. However, these companies are probably much larger targets for hackers than your company. And it's hard to protect against a virus that's completely new. So, it's really not a simple task to estimate your risk.

Like anything new, it will take time to determine the precise characteristics of the cloud. It will probably take five to ten years for the market to mature to a point where the uncertainties are small, the weak companies disappear and standards are set. For example, suppose you are late with a payment. Can the company withhold your data and cripple your company? What happens if they go bankrupt? There are many nightmare scenarios that are limited only by the imagination. Of course, it's hard to say what's realistic and what isn't.

Change in Infrastructure

However, there is a somewhat subtle aspect that is not always considered when employing Cloud Computing. Fundamentally, the company is out-sourcing part of the Information Technology (IT) Department. This may be good or not. It depends on the company and the IT Department. The problem is that the out-sourcing is not obvious. What's more, it starts the company down a path that is difficult to reverse.

When a company out-sources product assembly or accounting, the action is clear. Work is being shifted from inside the company to a different company. But with the cloud, the IT department just orders some new software. Everything seems the same. Yet, as time passes, more and more applications and data are being quietly shifted off-site. After a while, the on-site hardware and software requirements are significantly reduced. The IT department will shrink and personnel may be let go for lack of work. Quite possibly, the IT department will morph into a service department that interfaces the company to the cloud supplier. Again, this may or may not be good for the company. The important point is that all of the consequences of moving to the cloud need to be foreseen and considered before any action is taken. And it seems that it is often the case that the IT department sows the seeds of its own dissolution without realizing it.

Further, you can now see that trying to extract yourself from the cloud is not an easy thing to do. You will have to invest heavily in hardware and software, hire and re-train IT professionals and basically re-invent the wheel. Once you're in the cloud it's hard to get back to earth.

(130182)

Limited Time Offer for GREEN and GOLD Members!
13% DISCOUNT + FREE SHIPPING
www.elektor.com/june

OS Hard- and Software for Electronics Applications

1 Open Source Electronics on Linux

If you have ever wanted to take advantage of the expanding field of open source software for electronics and everyday applications, this book is for you. Using the Linux OS, Warwick A. Smith guides you through the world of open source hardware and software, teaching readers to use EDA tools and software that is readily available online, free to download. The hardware projects inside can be built using easily obtainable parts, in the comfort of your own home, on single sided PCBs, or professionally manufactured with output files generated by you. Open Source Electronics on Linux is about changing today's electronics enthusiast into empowered, savvy, discerning engineers capable of building and modifying their creations, be it solely on Linux or in tandem with your current operating system.

272 pages • ISBN 978-1-907920-19-6
$47.60

Programming step-by-step

2 Android Apps

This book is an introduction to programming apps for Android devices. The operation of the Android system is explained in a step by step way, aiming to show how personal applications can be programmed. A wide variety of applications is presented based on a solid number of hands-on examples, covering anything from simple math programs, reading sensors and GPS data, right up to programming for advanced Internet applications. Besides writing applications in the Java programming language, this book also explains how apps can be programmed using Javascript or PHP scripts. When it comes to personalizing your smartphone you should not feel limited to off the shelf applications because creating your own apps and programming Android devices is easier than you think!

244 pages • ISBN 978-1-907920-15-8
$56.40

A whole year of Elektor magazine on a single disk

3 DVD Elektor 2012

The year volume DVD/CD-ROMs are among the most popular items in Elektor's product range. This DVD-ROM contains all editorial articles published in Volume 2012 of the English, American, Spanish, Dutch, French and German editions of Elektor. Using the supplied Adobe Reader program, articles are presented in the same layout as originally found in the magazine. An extensive search machine is available to locate keywords in any article. With this DVD you can also produce hard copy of PCB layouts at printer resolution, adapt PCB layouts using your favorite graphics program, zoom in / out on selected PCB areas and export circuit diagrams and illustrations to other programs.

ISBN 978-90-5381-273-0 • $37.90

Display, buttons, real time clock and more

4 Elektor Linux Board Extension

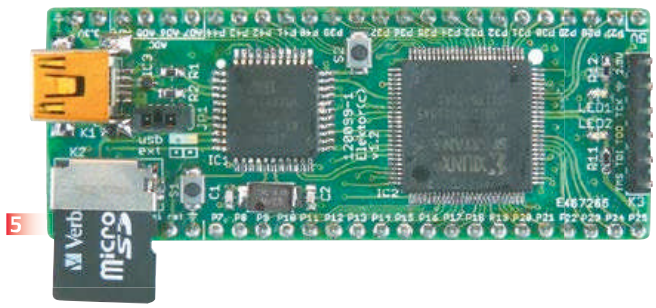
This extension board was developed to further propel our Embedded Linux series of articles and the matching GNUblin board. It has a display, buttons, a real time clock and 16 GPIOs. Linux devotees, switch on your solder irons. The Linux extension board includes everything needed to provide the user interface for a wide variety of projects!

Module, SMD-populated and tested board, incl. LCD1, X1, K1-K4, BZ1, BT1 for home assembly
Art.# 120596-91 • $50.20

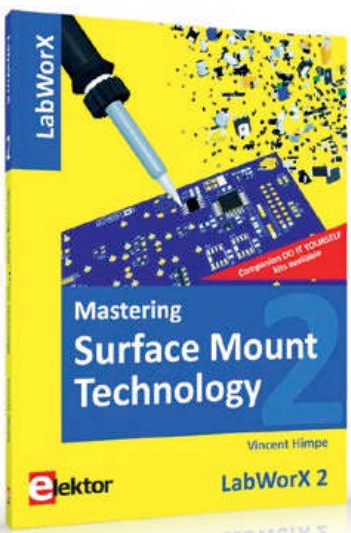
Taming the Beast

5 FPGA Development Board

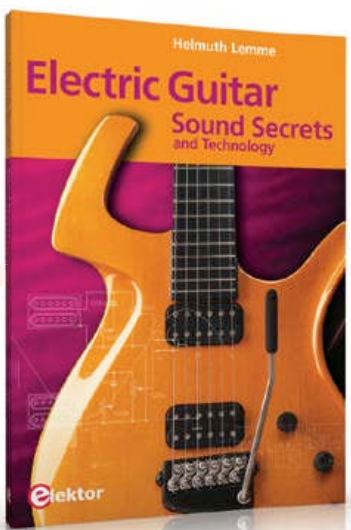
FPGAs are unquestionably among the most versatile but complex components in modern-day electronics. An FPGA contains a maze of gates and other circuit elements that can be used to put together your own digital circuit



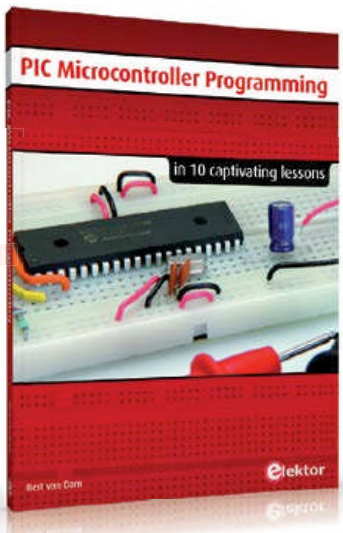
5



6



7



8

on a chip. This FPGA development board (designed in the Elektor Labs) shows how easy it is for any electronics enthusiast, whether professional or amateur, to work with these programmable logic devices.

Module, ready build and tested Art.# 120099-91
See www.elektor.com/fpgaboard

LabWorX 2

6 Mastering Surface Mount Technology

This book takes you on a crash course in techniques, tips and know-how to successfully introduce surface mount technology in your workflow. Even if you are on a budget you too can jumpstart your designs with advanced fine pitch parts. Besides explaining methodology and equipment, attention is given to SMT parts technologies and soldering methods. Many practical tips and tricks are disclosed that bring surface mount technology into everyone's reach without breaking the bank. A comprehensive kit of parts comprising all SMT components, circuit boards and solder stencils is available for readers wishing to replicate three projects described in this book.

282 pages • ISBN 978-1-907920-12-7
$47.60

Sound Secrets and Technology

7 Electric Guitar

What would today's rock and pop music be without electric lead and bass guitars? These instruments have been setting the tone for more than forty years. Their underlying sound is determined largely by their electrical components. But, how do they actually work? This book answers many questions simply, in an easily-understandable manner. For the interested musician (and others), this book unveils, in a simple and well-grounded way, what have, until now, been regarded as manufacturer secrets.

The examination explores deep within the guitar, including pickups and electrical environment, so that guitar electronics are no longer considered highly secret. With a few deft interventions, many instruments can be rendered more versatile and made to sound a lot better – in the most cost-effective manner.

287 pages • ISBN 978-1-907920-13-4
$47.60

10 captivating lessons

8 PIC Microcontroller Programming

Using the lessons in this book you learn how to program a microcontroller. You'll be using JAL, a free but extremely powerful programming language for PIC microcontrollers. Assuming you have absorbed all lessons you should be confident to write PIC microcontroller programs, as well as read and understand programs written by other people. You learn the function of JAL commands such as include, pin, delay, forever loop, while loop, case, exit loop, repeat until, if then, as well as the use of functions, procedures and timer- and port interrupts. You make an LED blink, build a time switch, measure a potentiometer's wiper position, produce sounds, suppress contact bounce, and control the brightness of an LED. And of course you learn to debug, meaning: how to spot and fix errors in your programs.

284 pages • ISBN 978-1-907920-17-2
$47.60

Further information and ordering:

www.elektor.com/store

Elektor US
111 Founders Plaza, Suite 300
East Hartford, CT 06108
USA
Phone: 860.289.0800
Fax: 860.461.0450
E-mail: order@elektor.com

Next month in Elektor

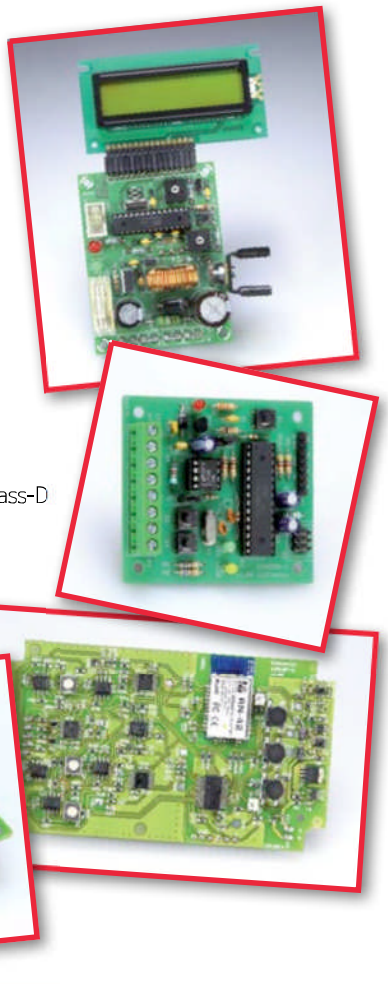
Elektor Project Generator Edition 2013

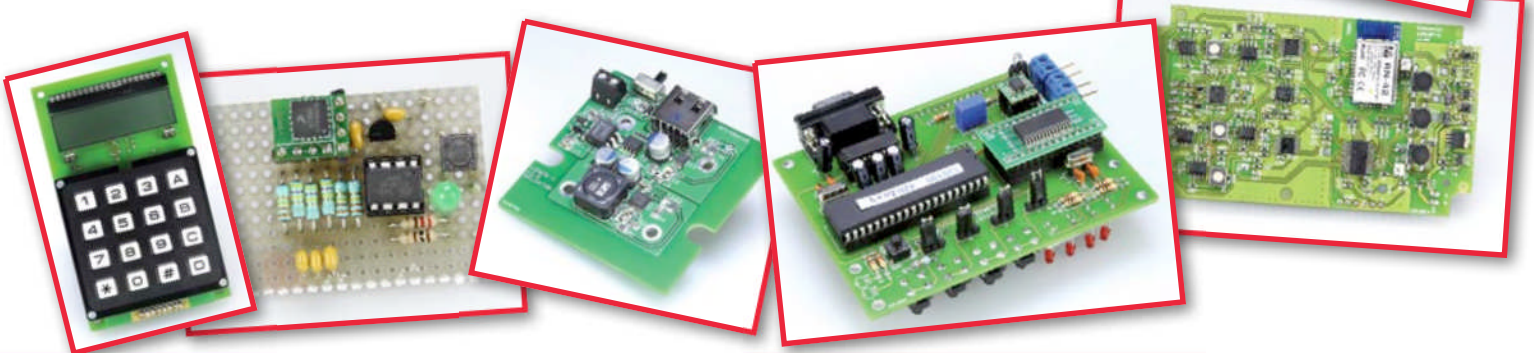
Extra-thick edition with an extra bunch of circuits

Next month we publish the 37th edition of Elektor's famed double edition for the months of July and August. This loved & collected issue offers a bulk of ideas, projects and practical software in a mix of small and large articles. Our editors and designers are now burning their midnight oil to make sure you can enjoy lots of pages with detailed descriptions and inventive electronics applications.

Don't miss it, this jumbo edition of Elektor packed with articles and projects like:

CAN-checker – ElektorBus-steppermotor control – CDI-ignition for mopeds – Inrush current limiter – Class-D audio amplifier with 555 – Servo checker – ECG-monitor for Android – USB-power pack – Numitron Arduino clock – Parking help – Multi-channel temperature logger – IR remote for Android – Extremely accurate digital clock – Universal precision measurement interface – AVR-USB-keyboard-stick





Article titles and magazine contents subject to change; please check www.elektor-magazine.com

Elektor July & August 2013 edition published June 26, 2013

See what's brewing @ Elektor Labs 24/7

Check out

www.elektor-labs.com

and join, share, participate!

ORDERING INFORMATION

To order contact customer service:

Phone:	860.289.0800
Fax:	860.461.0450
Mail:	Elektor US
	111 Founders Plaza, Suite 300
	East Hartford, CT 06108
	USA
E-mail:	order@elektor.com
On-line at	www.elektor.com/store

Customer service hours: 8:30 AM-4:30 PM EST Monday-Friday. Voice mail available at other times. When leaving a message please be sure to leave a daytime telephone number where we can return your call.

PLEASE NOTE: *While we strive to provide the best possible information in this issue, pricing and availability are subject to change without notice. To find out about current pricing and stock, please call or email customer service.*

COMPONENTS

Components for projects appearing in Elektor are usually available from certain advertisers in the magazine. If difficulties in obtaining components are suspected, a source will normally be identified in the article. Please note, however, that the source(s) given is (are) not exclusive.

PAYMENT

Orders must be prepaid. We accept checks or money orders (in US $ drawn on a US bank only), VISA, Mastercard, Discover, and American Express credit cards. We do not accept C.O.D. orders. We also accept wire transfers. Add $20 to cover fees charged for these transfers.

TERMS OF BUSINESS

Shipping Note: All orders will be shipped from Europe. Please allow 3–4 weeks for delivery. Shipping and handling via airmail: $20.00 per order.

Returns

Damaged or miss-shipped goods may be returned for replacement or refund. All returns must have an RA #. Call or email customer service to receive an RA# before returning the merchandise and be sure to put the RA# on the outside of the package. Please save shipping materials for possible carrier inspection. Requests for RA# must be received 30 days from invoice.

Patents

Patent protection may exist with respect to circuits, devices, components, and items described in our books and magazines. Elektor accepts no responsibility or liability for failing to identify such patent or other protection.

Copyright

Limitation of liability

Elektor shall not be liable in contract, tort, or otherwise, for any loss or damage suffered by the purchaser whatsoever or howsoever arising out of, or in connection with, the supply of goods or services by Elektor other than to supply goods as described or, at the option of Elektor, to refund the purchaser any money paid with respect to the goods.

MEMBERSHIPS (US & CANADA ONLY)

Order memberships on-line at www.elektor.com/members

All memberships begin with the current issue. Expect 3–4 weeks for receipt of the first issue.
Membership renewals and change of address should be sent to:

Elektor US
P.O. Box 462228
Escondido, CA 92046

E-mail: elektor@pcspublink.com

Memberships may be paid for by check or money order (in US $ drawn on a US bank only). We accept Mastercard, VISA, Discover and American Express credit cards.

For gift memberships, please include gift recipient's name and address as well as your own, with remittance. A gift card will be sent on request. Memberships may be cancelled at any time for a refund of all unmailed issues.

Does your membership expire soon?

Renew it on-line at www.elektor.com/members