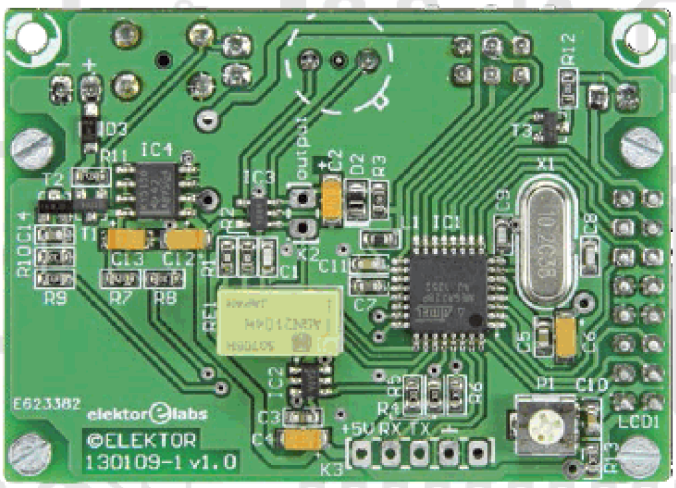
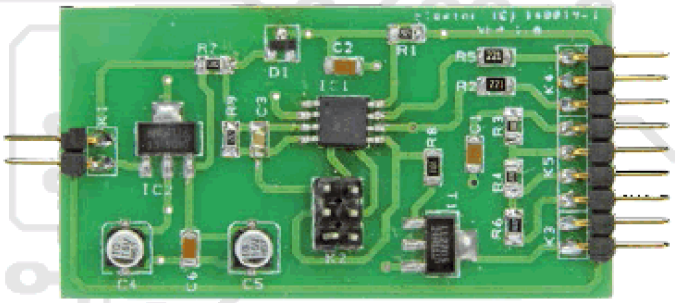
www.elektor-magazine.com

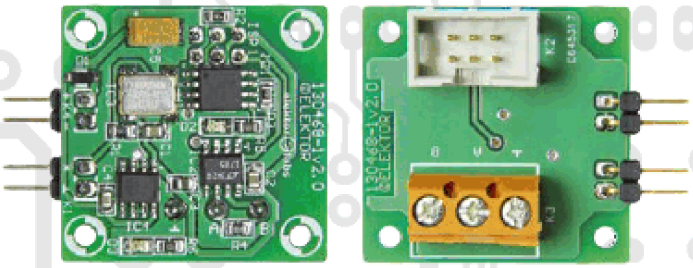
.magazine

October 2014

elektor

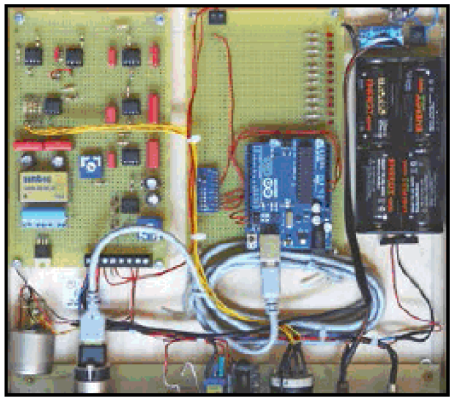


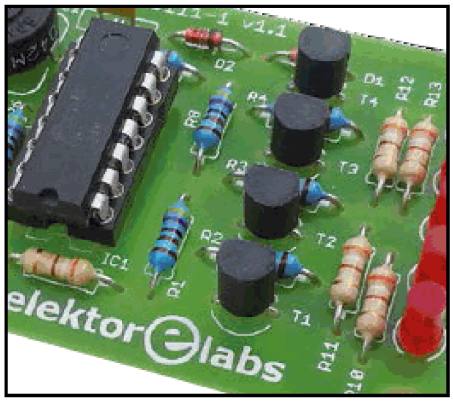


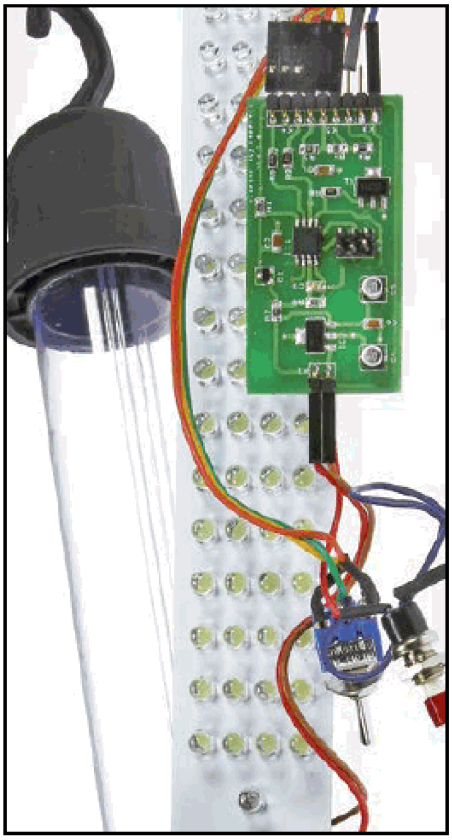


Extremely Low Frequency (ELF) Receiver | PWM Control for Flashlight Temperature Sensor Board | **Dot Display Driver** | **Microcontroller BootCamp (6)** **IoT & the Search for a Protocol** | Lux Meter | Chip Tip: MagI³C-VDRM Visual Basic on the Raspberry Pi | **DesignSpark Tips & Tricks** | **Weird Components: Magnetrons** **3D Printing Sure Can Be Useful** | USB Fix ● Review: Atmel ICE Debuggger/ Programmer ● Retronics: a 1965 Telefunken Carphone

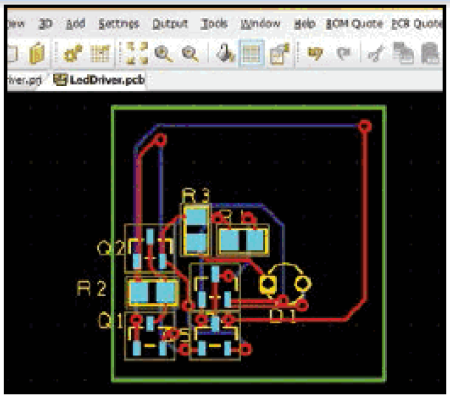
elektor magazine

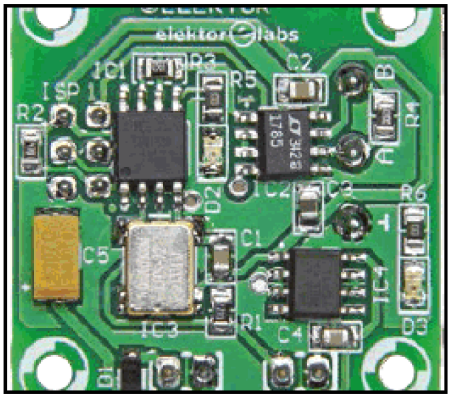


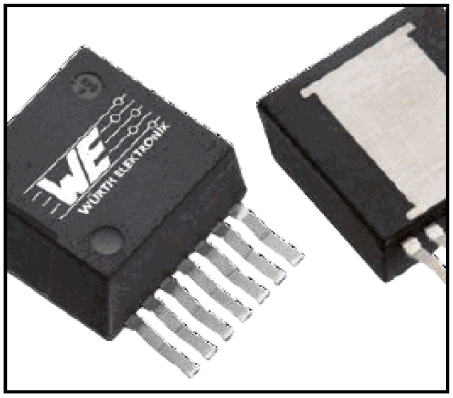


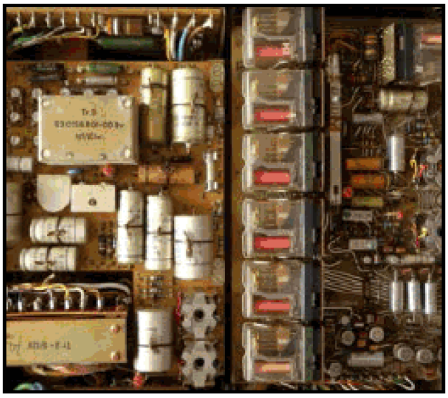


Projects









Review

Industry

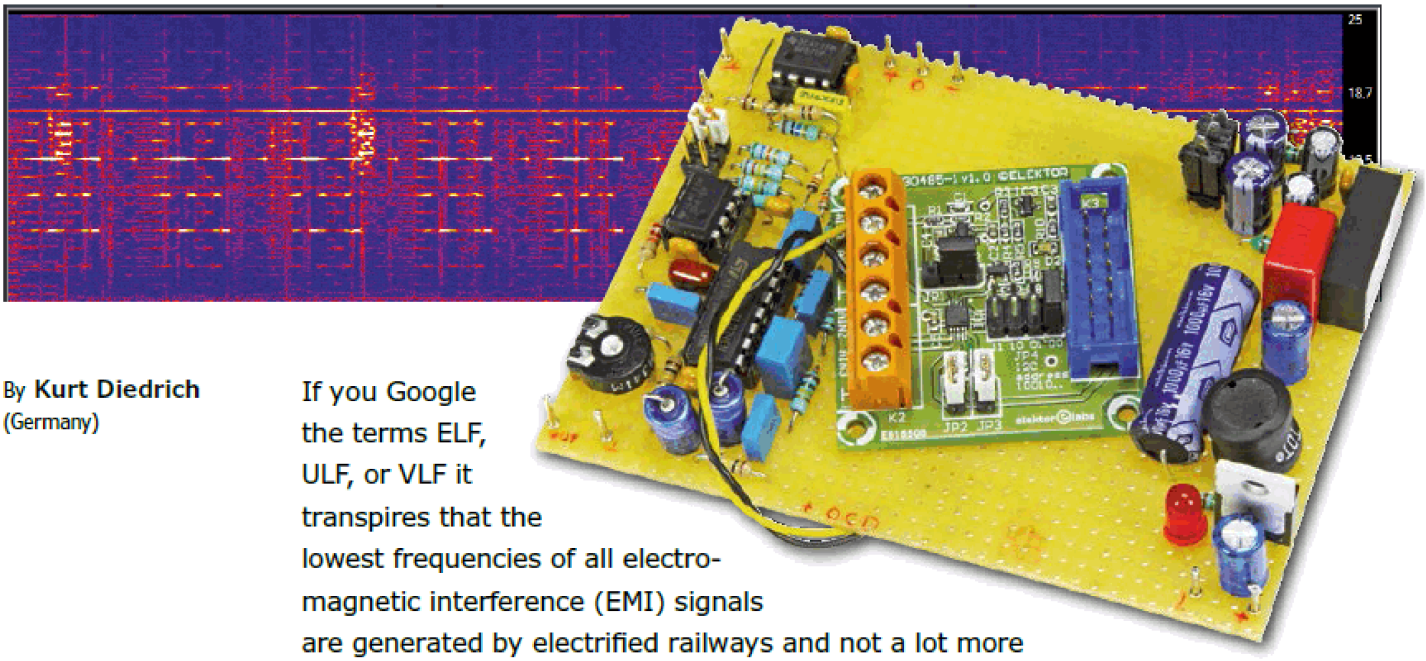
DesignSpark

Labs

Regulars

Extremely Low Frequency (ELF) Receiver

Arduino + ADC = ELF



By **Kurt Diedrich**
(Germany)

If you Google the terms ELF, ULF, or VLF it transpires that the lowest frequencies of all electromagnetic interference (EMI) signals are generated by electrified railways and not a lot more beyond this. Wrong! You can in fact receive some extremely interesting signals between 0 Hz and the 'railway' frequency of $16^2/_3$ Hz. Using the receiver described here with an ADC module described in a separate article and some free PC software it is possible to receive and make recordings of these signals.

It had always fascinated me what I might hear—or rather see on an oscilloscope—if I could connect a pick-up coil, with a couple of hundred turns on it, to an extremely sensitive amplifier. A dozen or so years ago I decided to turn this supposition into fact using modern electronics.

The first circuit I constructed for this purpose differed from the version presented here only by having a cruder filter and a somewhat older-fashioned method of analog to digital conversion. To my surprise there appeared on the monitor screen more than the power frequency hum that I was expecting but unfortunately the confused serrations of the complex time signals did not allow me to draw any conclusions from about their composition. [this article was written in Europe where the AC supply frequency is 50 Hz but exactly the same methods will work in territories where the line frequency is 60 Hz. Please read '60' wherever you see '50' from now on, if you live in a 60 Hz country. *Ed.*]

Eventually, after I submitted my received data to FFT-versus-Time analysis, it became very clear to me that that this 'wriggling about' on the screen was the result of recurring signals of typical structures, which could be resolved only if they could be compressed over prolonged periods of time. They were also audible if played back at higher speed, sometimes reminiscent of animal sounds or teletype transmissions on the short waves. In any case, all this was sufficiently interesting to keep me occupied with it ever since. Readers who

are interested will find further detailed information at the blog vlf.it [3], which is a platform for enthusiasts involved with receiving and experimenting in the ELF and VLF bands. I have published a number of articles there on this theme along with many screen shots.
Among other things, we need to understand that supply transformers in residential areas radiate extremely weak magnetic waves between around 0.3 Hz and 25 Hz. These are up to 1,000 times weaker than the interference fields produced by the 50 Hz AC supply. To receive the desired frequencies without interference, we need to filter out the 50 Hz (60 Hz) supply hum as early as possible ahead of the main amplifier in order to avoid over-driving the receiver.

The circuit

The receiver described here operates in conjunction with the ADC module described in a separate article, an Arduino Uno and some free—that goes without saying—recorder software for the PC. This combination makes it possible to detect, display and log weak alternating currents and/or alternating magnetic fields at frequencies down to less than 1 Hertz. The receiver output can additionally be connected to other recording devices, all the time keeping in mind that signals below 16 Hz will be attenuated heavily by PC sound cards.
The circuit is made up from a combination of a highly sensitive voltage amplifier and a steep (36 dB per octave) Sallen Key low-pass filter with a cut-off frequency of approximately 21 Hz. The receiver has the task of amplifying extremely weak magnetic waves in the frequency range from 21 Hz down to (almost) 0 Hz and filtering out line hum interference in the process. **Figure 1** shows the schematic for this receiver, which is made up from the functional groups that follow.

Linearizer and preamp

The extremely weak (in the microvolt region) AC signals of interest here are picked up with a coil and once processed and optimized in a combination of preamplifier and low-pass filter (IC1), they are directed to the Sallen Key low-pass filter that follows. This simple upstream low-pass filter (a pre-filter so to speak) is necessary specifically for attenuating any 50 Hz line frequency interference in relation to the wanted signal to prevent overloading that might generate a square-wave signal between the maximum output voltages

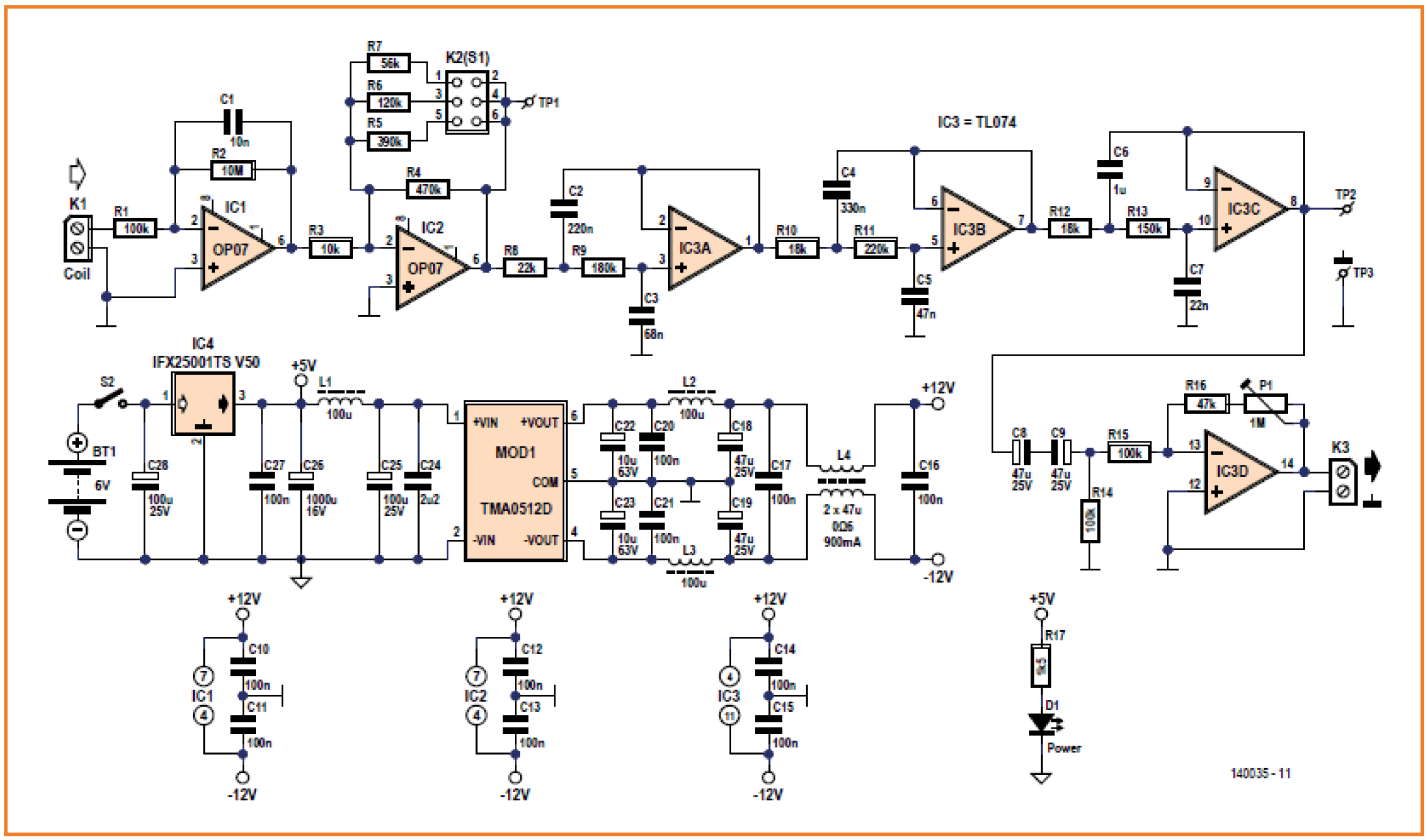


Figure 1.
Schematic of the ELF Receiver (without Data Logger).

of the op-amp. This could occur were the coil to be placed close to a power cable in which a heavy current was flowing. The circuitry associated with IC1 has a second function: the characteristics of the coil at the input of the circuit mean that low frequencies are attenuated appreciably, so that the amplitude of received signals in the region of zero Hz is weakened increasingly. We can compensate or 'linearize' this to a large extent using the effect of capacitor C1 in parallel with R2. **Figure 2** shows the amplification at the output of IC1.

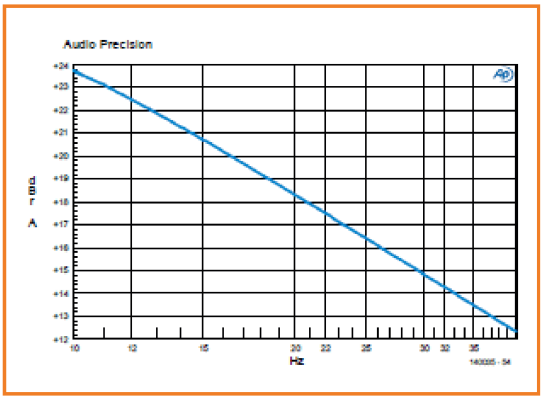


Figure 2.
Amplifier flatness at the output of IC1.

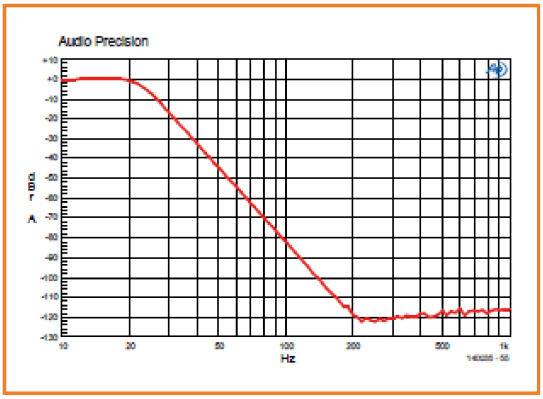


Figure 3.
Filter IC3 achieves a slope of around 36 dB per octave!

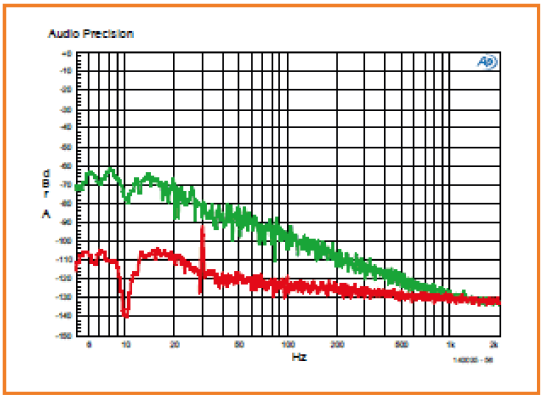


Figure 4.
The common-mode choke reduces interfering noise by up to 40 dB.

The stage built around IC2 is a well-known 'standard' circuit involving an inverting amplifier. The gain factor can be varied by selecting one of several feedback resistors. This feature is absolutely necessary since completely different intensities of the received signal may arise, according to the position of the receiver. R4 is not a built-in part of the selector switch, ensuring that there is always some degree of negative feedback, even when the switch settings are open-circuit. This has the advantage that at the moment of switchover, when the switch contact 'hangs in mid-air' for a very brief timespan, no interference pulses appear on the receiver output.
The gain or amplification of the inverting amplifier arises from the quotient negative feedback resistance divided by the upstream resistor:

$V = R_g/R_v$

By switching from R7 down to R4 alone we have successively (approximate) gain settings of 5, 10, 21 and 47, the last of these values being when the three switches or jumper links are all open-circuit.

Filter

The remaining four op-amps are combined in a single IC, the TL074. IC2A to IC2C together form a Sallen Key Filter with fast roll-off providing 36 dB per octave in total. The elevated level of the 50 Hz signal relative to the desired signal makes this filter extremely necessary, to prevent overloads. To learn more about Sallen Key Filters you can find the desired background information in the technical literature and on the Internet [2]. The cut-off frequency of the filter is, precisely stated, 21.5 Hz, which is far enough removed from the interfering 50 Hz and is still outside the desired reception range.
You should stick to the values given for the capacitors and resistors as closely as possible, as the required transfer characteristic cannot be guaranteed. **Figure 3** shows how steep the flanks of the resulting filter are (measured at the output of IC3C).

High-pass and final stage

At high levels of gain (according to the setting of P1 up to about 50,000) it's possible that even quite small offset voltages could nevertheless be sufficiently large to shift the output signal by several volts into the positive or negative regions

and become an undesirable disruption. To avoid this, a high-pass filter (C8+C9/R14) is provided between the filter output (IC3C) and the input to the amplifier stage IC3D following, with a cut-off frequency arranged to lie well below the target range. In this way the filter does not affect the frequency response of the received signals. The voltage at the the output of IC3D is thus always symmetrical around zero. Trimpot P4 is used to adjust the total gain of IC3D between 0.5 and about 10.5. In conjunction with the switchable pre-filter stage this should suffice for well-nigh all user situations.

Powering the circuit

Power connections are provided for 6 V rechargeable (or plain) battery operation. Initially the voltage is reduced to 5 V and stabilized in IC4, to suit the needs of the TMA0512D converter that follows. This converter changes the input voltage of 5 V into two complementary output voltages of 12 V, used for powering the op-amps. Do not omit any of the chokes and capacitors shown in the schematic of the power supply section, as these are absolutely necessary to reduce interference from electrical noise. **Figure 4** shows, for example, the beneficial effect of using the common-mode choke L4; this suppresses noise in the relevant frequency range by around 30–40 dB!
If you prefer to power the receiver using an AC adapter rather than batteries you can connect a 6-V wall wart power adaptor of the necessary amps rating.

Coil and electrodes

To detect weak magnetic fields a sensitive receiving antenna is necessary, so we should connect a coil with around 2,000 to 4,000 turns of as large a diameter as possible. This does not have to be as full-blown an affair as the one shown in **Figure 5**; a diameter of 12 to 20 inches (30 – 50 cm) will be perfectly adequate (to begin with). The sensitivity of the coil (not to be confused with its inductance!) increases linearly with the area enclosed by the coil former and the number of turns. The coil should be ring-shaped and if you buy the wire from a specialist supplier coiled in a roll [3], you can create your coil rapidly and easily using a coil-winding machine made at home from an old *Erector* or *Meccano* outfit.
Enameled copper 'magnet' wire of 30 AWG (0.25 mm) diameter turned out to be a particularly good choice for making the coil. It is not so thin that it would snap instantly if handled roughly.
During signal reception the coil must lie flat on a non-metallic surface, as far away as possible from any AC power cables with current flowing through them. Important: on account of the Earth's magnetic field, the receiver should be operated only when the coil is not subjected to any movement or agitation.
As an alternative to the pickup coil, the receiver can also be used with electrodes, consisting of metal probe spikes about 8 inches (20 cm) long, pushed into the ground at a distance about 7 feer (2 m) apart. In this way you can detect alternating currents in the prescribed frequency range present in the Earth's surface.

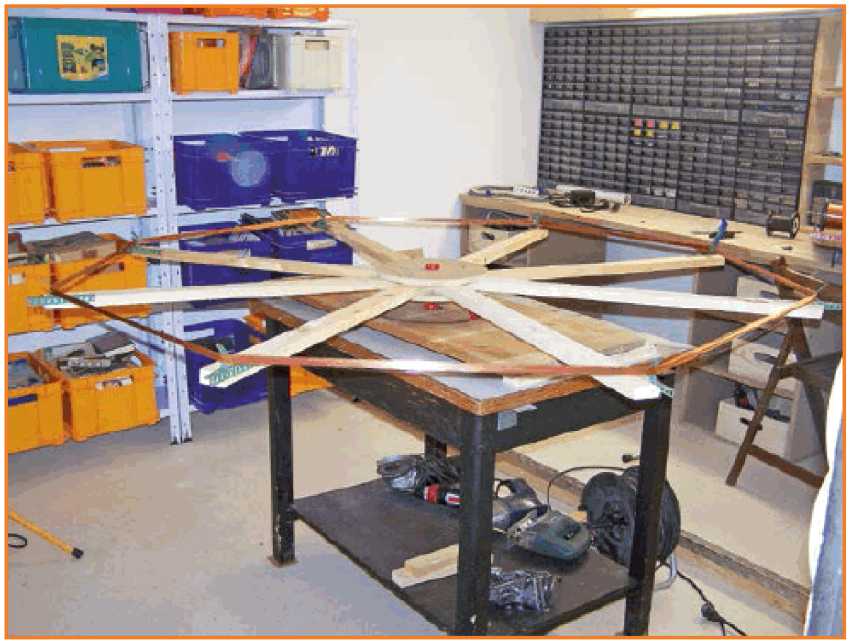


Figure 5.
The author's home-made coil winding machine. Smaller versions will suffice for practical applications.

Safety warning: If you are working with ground electrodes bear in mind the risk of rogue AC power voltages in the soil. For this reason it is vital to use a 1:1 microphone transformer (isolating transformer) on the input of the receiver whenever the receiver (or any other device connected to it) is powered from the AC supply. I have experimented with an example made by the firm *Jensen* that has proved to be absolutely ideal (type JT-11P-1). A variety of suitable types are shown on this American firm's website [4].
Use unscreened cable to connect the ground spikes to the transformer, the secondary side of which is hooked up to the receiver input in place of the coil.

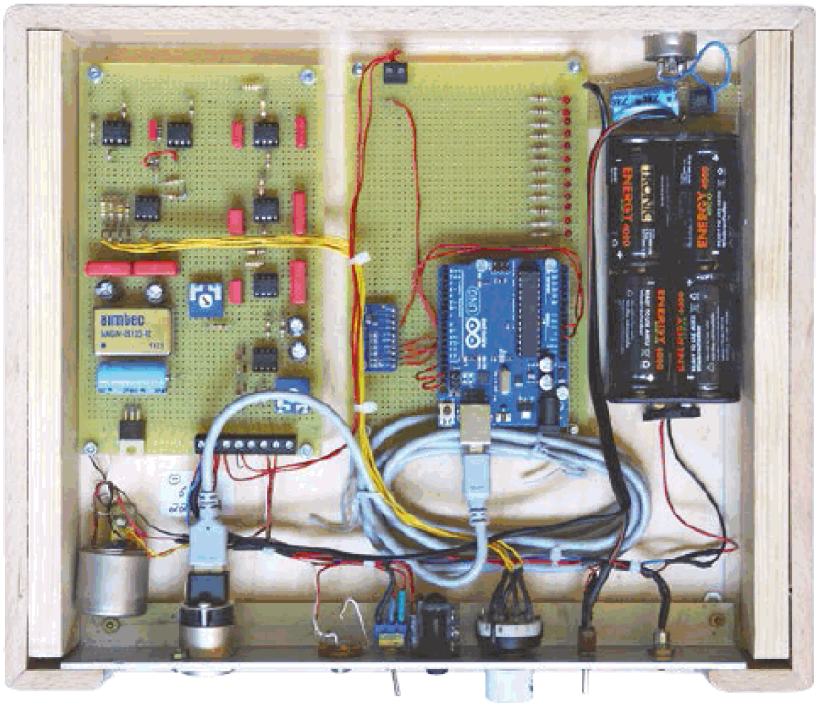
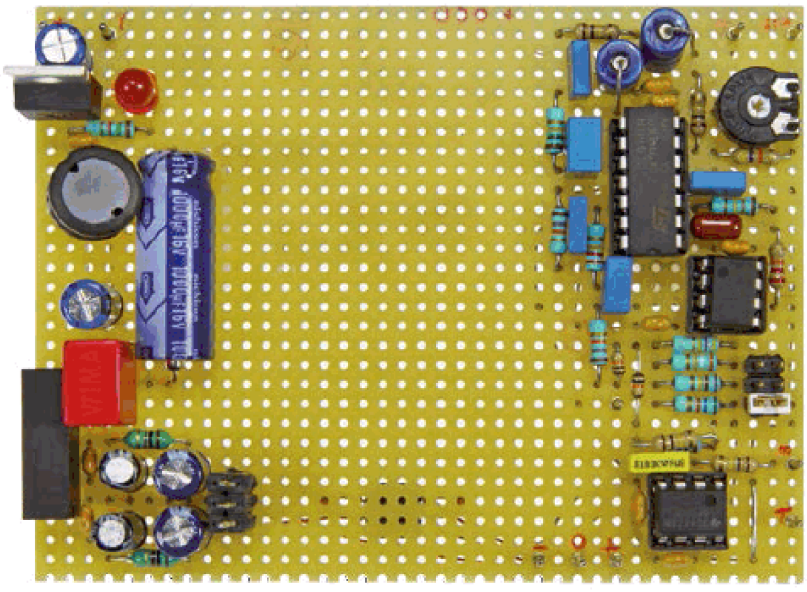


Figure 6.
The author's first prototype. The Arduino data logger is also visible in the case.

Figure 7.
Test build signed off by Elektor Labs.



Alignment, connection and testing

A printed circuit board layout is not provided for the circuit of the ELF receiver, so readers interested in replicating it must resort to self-help or simply assemble the small number of components involved on a piece of perf-board or strip board (Vectorboard; Veroboard). Programs like *LochMaster* from Abacom or the free *Blackboard* [5] will be of assistance for laying out the board.

The author's prototype can be seen in **Figure 6**, with **Figure 7** showing the test build made with stripboard in Elektor Labs.

Once you have finished the construction and testing, the alignment of the receiver with the oscilloscope can begin. Hook the coil up to the input, arm yourself with a strong magnet (such as one from a loudspeaker) and investigate each of the outputs of the op-amps in succession. At the output of IC2 the line hum (so far only pre-filtered and receivable everywhere) should not exceed the 50 % overload limit. If the options available at K2/S1 are insufficient for this, then the value of R3 should be increased.

As you investigate each successive IC output, the 50 Hz (60 Hz) sinewave components should become ever weaker. Now set the oscilloscope to 1 V/Div and move the magnet to and fro by hand at a distance of two meters from the coil, once or twice a second. You should now be able to observe clear deflections up to the clipping limit. It should also be possible to detect a slight ripple even without moving the magnet, resulting from ambient signals (unless your home is in the middle of a forest). Next adjust P1 so that the peak values of this ripple amount to no more than ±1 V and lie within the optimum range of the A-to-D converter. The output signal should be free of any offset voltages whatsoever. Also the line frequency sinewave oscillations should be barely detectable now. At output K3 you should now have a pure AC signal for downstream processing by the ADC.

Installation and operation of the recorder

Now connect the ELF receiver to the 16-bit Datalogger module described in the September 2014 edition of Elektor [13]. The ADC samples the signal with a resolution of 15 bits and a sampling rate of around 112 Hz. The article also explains how the ADC module can be connected to an Arduino Uno, which accepts the digitized data using a simple program (*Sketch*) and relays this to the PC and the recording software.

The recorder software is written in the *Processing* programming language [12], which resembles *C*. Curly brackets are used for code blocks; each instruction must be closed off with a semicolon. The programming environment is very simple: just open the Editor and write the source text. Then click on the Start button and you're rolling.

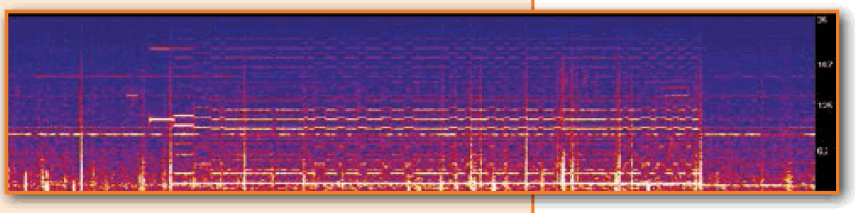
What is ELF?

ELF signals are a mysterious and, to some degree, myth-ridden subject that amounts in reality to nothing more than electromagnetic waves of extremely low frequency (hence ELF) from 3 Hz to 30 Hz. Because commercial radio transmissions do not exploit such low frequencies, it is naturally fascinating to investigate what is going on in this profound realm.

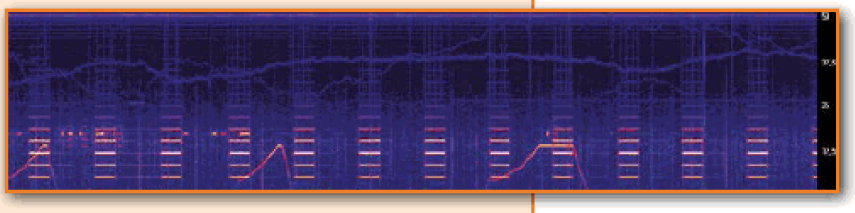
In residential areas many of the signals detectable with the receiver described here clearly take the form of magnetic waves radiated by supply transformers at the local substation. The sprawling network of metallic conductors (ground connections, water and gas pipes, etc.) evidently behave like a vast underground antenna that gathers up the weakest low-frequency alternating currents flowing in the ground, wherever they may arise from, and transports them to a common connection point at the local substation. Here (this is merely an assumption) these currents are radiated as magnetic fields by the grounded Petersen Coil (used for ground/earth leakage compensation).

In addition to these signals, previous considered indeterminate, we must note the increasing level of (mainly daytime-only) 'pollution' coming from (presumably) commercial and communal installations such as inverters, frequency changers and switch-mode power supplies. With a little patience it is also feasible to prove the presence of the fascinating so-called Schumann Resonances [11] in the region around 7.5 Hz along with the 16 $^{2}/_{3}$ Hz (50 Hz ÷ 3) 'signature' of traction current used by electric trains, which makes an excellent marker signal for testing and calibrating the receiver (in places where traction current uses this frequency). Another conceivable application (somewhat frivolous in comparison) for the receiver is as a highly sensitive detector for (exclusively) moving metallic objects. Passing automobiles, for example, can be detected at distances of up to around 20 meters or 60 feet.

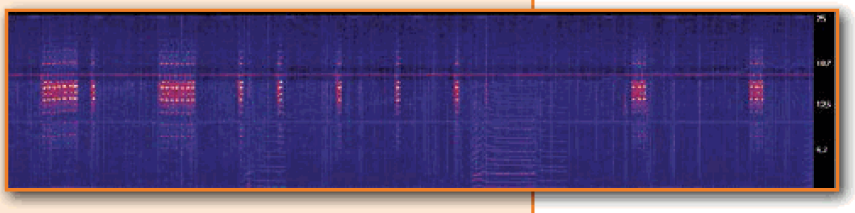
The following shots show examples of some varied and interesting 'signal harvests':



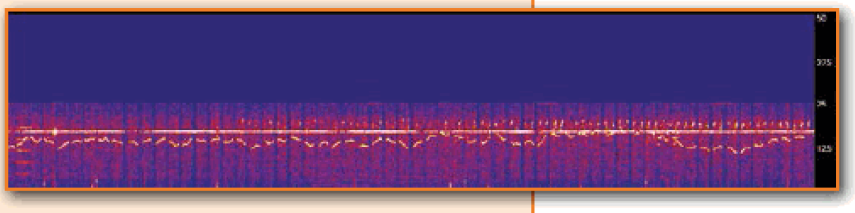
Wow! This signal occurred on one single occasion over a night in September 2013. Duration around one hour. Recording made with electrodes. Frequency range: 0 to 20 Hz.



A square-wave signal of 1.6 Hz, which arises in various locations across all Europe at irregular times. Typical characteristics: phases of activity and intervals changing regularly.



Extremely powerful 16 Hz bursts, concentrated at particular locations and even audible direct as a deep hum in audio amplifiers.



Sounding like whistling when played back at high speed, this sample has reappeared daily for several hours at the author's place of residence over the years.

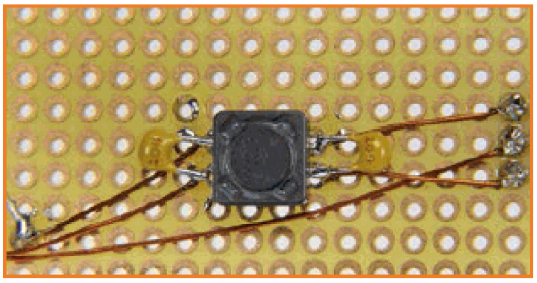


Figure 8.
Power section and filter/amplifier should be placed as far apart as possible from one another. Inductance L4 is fitted in between them (here on the underside of the PCB).

Software installation
To get a Processing program to run on your computer, you need to download the necessary software from the Internet onto your machine.
Go to the Processing website [7] and follow the instructions given there. The data downloaded can go into any folder you choose on the hard disk. Within this data is also a file with the name *processing.exe.* Run this program if you want to write Processing software of your own. Numerous impressive sample programs not only showcase the powerful capability of this language but also indicate how you can make the best use of it. The *Recorder* program written in Processing can be downloaded from the Elektor website [8] into any folder of your choice.

Important: The Processing program must be located in a sub-folder bearing the same name as the program itself—but without the '.pde' suffix. Also all resources required by the program (such as .wav files or associated graphics) must be kept in this sub-folder. After double-clicking on the recorder file (Recorder_.....pde) the Processing editor window opens automatically and the program code is implemented.
In the following line you need to replace 'COM3' and enter the COM interface of the PC allocated by Arduino (see Device Manager in Windows):

```
serport = new Serial(this, "COM3", 115200);
```

Then save the program code with File → Save.
After a (single) click on the arrow at top-left in the Editor window, the program begins. The Editor window with the source code remains during this process on the screen (in the background). Unfortunately (and not for want of searching countless different sources) I have not managed to find a working .exe file for the program. Further information about Processing can be found in the Editor itself (Help → Reference) and on countless other Internet pages.

Operation
Operation of the software recorder in the Windows-style window (**Figure 9**) is virtually self-explanatory. The test results are shown in three windows on the left-hand side.

Time signal
An iteration takes five seconds. After the program starts a signal is always visible here, even if it is not being recorded—and after recording stops.

FFT vs. Time
Every x seconds (x depending on the value set when downsampling) a new line is plotted—even if no recoding is being made—and after recording stops.

Supervisory signal
After each iteration of 5 seconds, the highest amplitude of this time segment is indicated in the upper window.

The parameters for measurement and display are set on the right-hand side of the recorder:

Recording time
Length of the recording.

Downsampling
Zoom in the Y direction in order to see the lower frequencies better. Relates only to the FFT displayed and *not* to the recording.

FFT brightness
Renders the FFT displayed brighter or darker. Relates only to the FFT displayed and *not* to the recording.

FFT scrolling
Pages forwards and backwards through the analysis data displayed. Valid only for the data recorded during the current recording phase still held in RAM. FFT data is not stored on the hard disk.

Mouse position
Mouse position coordinates and number of buttons clicked. Very important if you wish to work on the program yourself.

Recording
Left-hand button
Normal method of starting a recording of a dura-

tion enter above, automatic saving at the end of the set time and regular saving intermediately. Show bright red during recording.
Note that the names of files saved automatically contain the start time and the intended stop time (for example 18:20, if the recording began at 12:20 and was set to record for six hours).
If the recording is cancelled ahead of time, you will find the previously saved data under file name planned at the outset (2014_07_02_1220_1820). The file can nevertheless contain just nulls rather than data from a particular point in the recording, according to the moment of premature cancellation. This is because in this programming language it is possible to save only complete Arrays and not, as is otherwise normal, only the section occupied with data. For this reason files always retain their full size, even if cancelled prematurely.

Center button
For intentional buffering. This has the same validity for the filename as with automatic saving. Pressing this button makes it possible to observe the data recorded up to the current time in an analyzer.

Right-hand button
This button cancels the recording and saves the data recorded so far to disk. Note that in this case, the stop time specified in the file name is the clock time valid at the actual time of cancellation. Pressing this button produces an additional file afterwards.

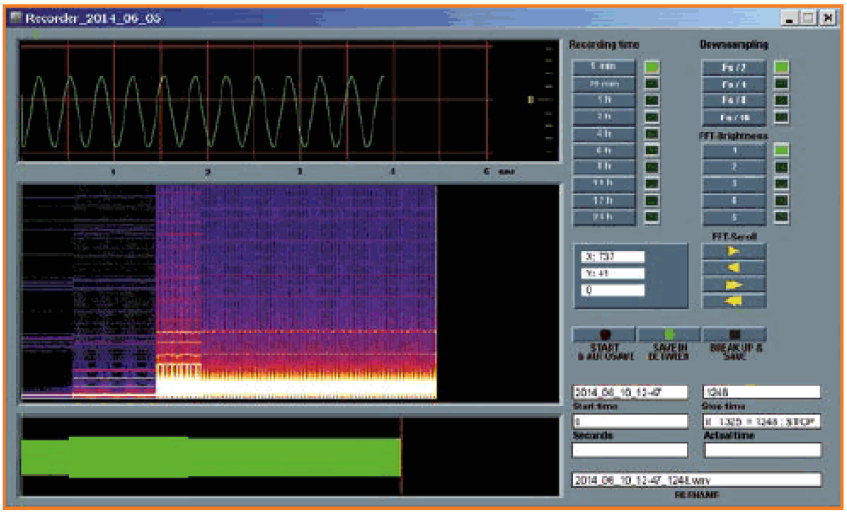


Figure 9.
Graphical interface of the recorder software.

Data output window
at lower right-hand edge of screen:
After starting, the time remaining until the time when the recording will end is displayed here automatically, based on the record duration set. At the very bottom is the file name, which is also retained during buffering.

(140035)

Web Links

[1] Online ELF blog: www.vlf.it

[2] https://en.wikipedia.org/wiki/Sallen%E2%80%93Key_topology

[3] Coiled wire: http://www.jameco.com/1/1/379-30pe-awg-plain-enamel-magnet-wire-1-4-lb-825-ft.html [USA], http://www.scientificwire.com/acatalog/Solderable_Enamelled_Copper_Wire.html Ref: SX0250s-D200 [UK]. Alternatively just look on eBay.

[4] Input transformer: http://www.jensen-transformers.com/In_in.html

[5] http://blackboard.serverpool.org/

[6] www.elektor-labs.com/project/arduino-16-bit-low-frequency-datalogger-130485-i-140035-i.13703.html

[7] http://processing.org/

[8] www.elektor-magazine.com/140035

[9] https://groups.yahoo.com/neo/groups/VLF_Group/info

[10] http://naturalradiolab.com/

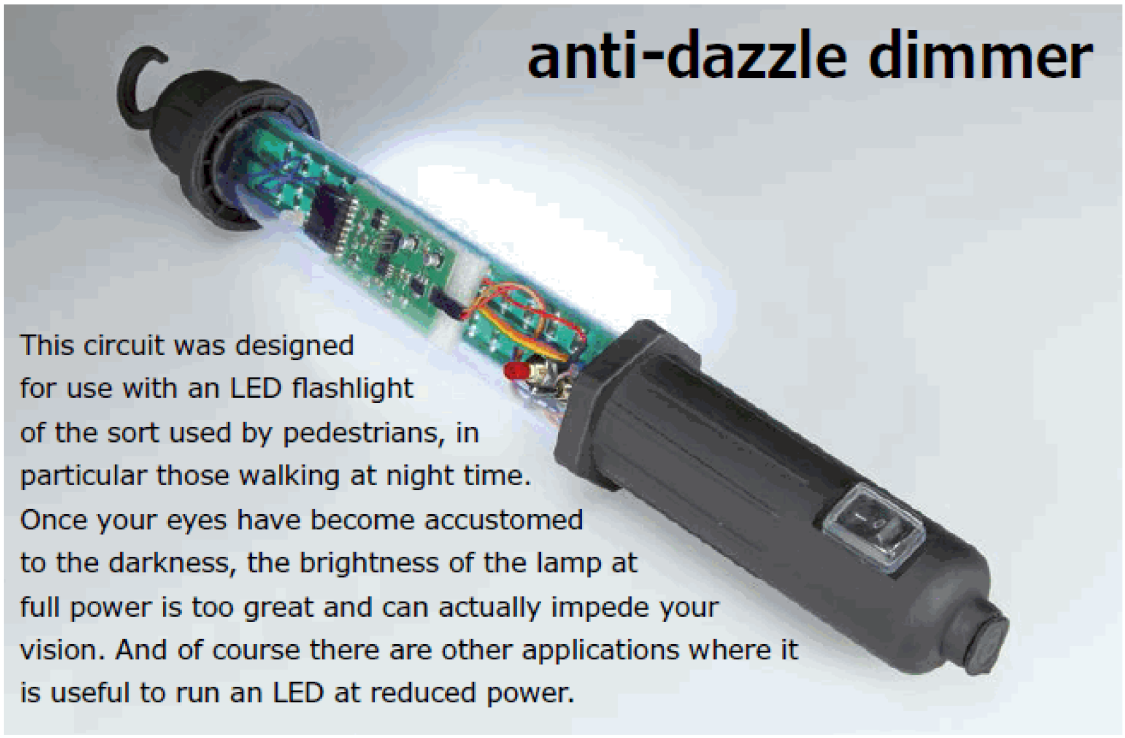
[11] http://en.wikipedia.org/wiki/Schumann_resonances

[12] http://en.wikipedia.org/wiki/Processing_(programming_language)

[13] www.elektor-magazine.com/130485

PWM Control for LED Flashlight

By
Pascal Rondane
(France)



This circuit was designed for use with an LED flashlight of the sort used by pedestrians, in particular those walking at night time. Once your eyes have become accustomed to the darkness, the brightness of the lamp at full power is too great and can actually impede your vision. And of course there are other applications where it is useful to run an LED at reduced power.

As its name suggests, the main function of this unit is to reduce the brightness of an LED at the user's command. An additional function is also provided: the LED can be flashed at full intensity, which can come in handy for example when you are walking at night, see an oncoming vehicle, and wish to alert the driver to your presence.
The circuit is based around a microcontroller which performs two tasks in parallel. First, it generates a PWM (pulsewidth modulation) signal to control the brightness of the LED according to which of the two modes is in force (steady light with reduced intensity or full-intensity flashing); and second, it monitors and displays the level of charge in the battery.
The PWM technique employed here to obtain a range of different brightness levels is convenient for use when modifying an existing flashlight, as well as being applicable more generally to LED lighting circuits powered from batteries. The dimmer circuit as it stands is designed for a battery voltage of 7.2 V but the regulator chosen is capable of accepting voltages up to 14 V or 15 V.

Features

- Supply voltage: 5 V to 15 V
- Current consumption: 3.9 mA (not including LED D2)
- Maximum output current: 1 A
- Battery state thresholds (adjustable)
  - Battery charged: 7 V
  - Battery low: 6 V
  - Battery flat (flashlight off): 5.4 V

The omnipotent microcontroller

The dimmer circuit (see **Figure 1**) is controlled by an 8-pin ATtiny45 microcontroller (IC1), which is in-system programmable.
The brightness of the flashlight's LED (shown on the right of the schematic inside the green dashed box) is controlled by the user by means of center-off momentary changeover switch S2.

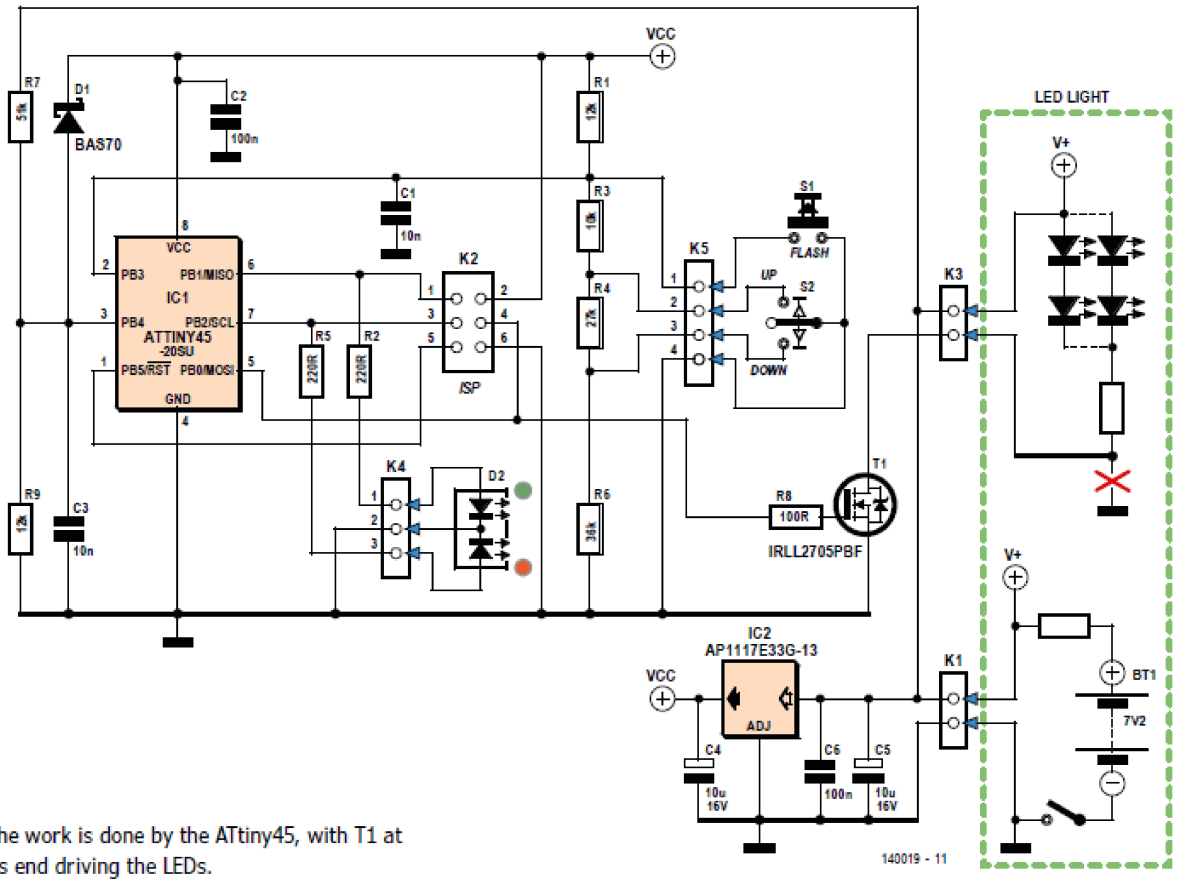


Figure 1.
Almost all the work is done by the ATtiny45, with T1 at the business end driving the LEDs.

Because the number of pins available on the ATtiny45 is so limited the state of the two controls on the dimmer unit is read using a single analog-to-digital converter input (pin 2) on the microcontroller. The voltage on this pin is obtained from the potential divider formed by R1, R3, R4 and R6, which are connected to switch S2 and pushbutton S1 via K5. When the user operates one of these controls, some of the resistors in the lower part of the potential divider are short-circuited, affecting the voltage on pin 1 of K5. From this voltage the microcontroller can determine which control was pressed and carry out the appropriate action: increasing or decreasing the brightness of the LED in the case of S2, or switching from steady mode to full-brightness flashing mode in the case of S1. The threshold voltages are as follows:

- 2.8 V: S1 and S2 both open
- 2.5 V: increase brightness ('UP')
- 1.5 V: decrease brightness ('DOWN')
- 0 V: flash

Connectors

- K1: power
- K2: in-system programming for the microcontroller
- K3: connection to the LED chain in the (modified) flashlight
- K4: battery charge status indicator
- K5: S1 and S2

The PWM output is also under software control. The corresponding pin on the microcontroller (pin 5) drives the LED chain in the flashlight via power MOSFET T1.
In order to avoid dazzling the user when the circuit is powered up, the circuit always starts with the LEDs at half brightness.
R7 and R9 form a further potential divider connected across the battery supply. The center tap of the divider is connected to an A/D input of the microcontroller (pin 3), allowing it to monitor

Component List

Resistors

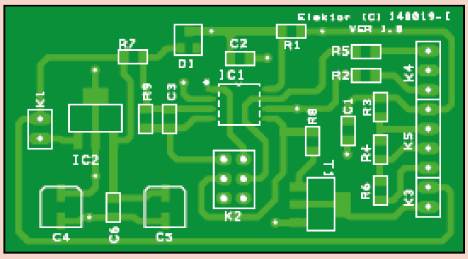
Default: SMD1206, 1%, .25W
R1, R9 = 12kΩ
R2, R5 = 220Ω
R3 = 10kΩ
R4 = 27kΩ
R6 = 36kΩ
R7 = 51kΩ
R8 = 100Ω

Capacitors

Default: SMD1206
C1 = 10nF*
C2, C6 = 100nF
C3 = 10nF
C4, C5 = 10µF 63V radial

Semiconductors

IC1 = ATtiny45-20SU (SOIC8)
IC2 = AP1117E33G-13 SOT-223)
D1 = BAS70* (SOT-23)



T1 = IRLL2705PBF (SOT-23)

Divers :

K1,K3 = 2-pin pinheader
K2 = 6-pin (2x3) pinheader
K4 = 3-pin pinheader
K5 = 4-pin pinheader
S1 = pushbutton
S2 = switch, on-off-on, center detent
PCB # 140019
*see text

Figure 2.
The printed circuit board is compact enough to fit inside a cylindrical-style flashlight. Nevertheless the surface mount components are sufficiently well spaced to allow manual soldering.

```
'#######################################################
'#  Battery voltage test (Voltage hysteresis = 0V)
'#######################################################

  Select Case Etat_led 'Voltage hysteresis = 0V

    Case 1 :    'Operating voltage => OK

      Tension_batt = Tension_batt_basse - Tension_hysteresis     '6V-0V = 6.0V
      If Adcval < Tension_batt Then 'ADC measurement and switching to low battery mode
        Etat_led = 2
      End If

    Case 2 :    'Low battery

      Tension_batt = Tension_batt_dechargee - Tension_hysteresis '5,5V-0V = 5.5V
      If Adcval < Tension_batt Then 'ADC measurement and go to discharged battery
              ' mode - cut the load
        Etat_led = 3
      End If
      Tension_batt = Tension_batt_chargee + Tension_hysteresis
      If Adcval >= Tension_batt Then
        Etat_led = 1
      End If

    Case 3 :    'If low battery/ cutoff LED
      Tension_batt = Tension_batt_chargee + Tension_hysteresis   '7,2V + 0V = 7.2V
      If Adcval => Tension_batt Then'ADC measurement and return
              ' "OK mode "if battery is charged
        Etat_led = 1
      End If

  End Select
```

Extract from the source code. The section shown deals with monitoring the battery status.

continuously the state of charge of the battery. This state is indicated using D2 as follows:

- **green** = battery charged;
- **orange** = battery low;
- **red** = battery almost flat.

It is not good for the battery to continue to discharge it when it is almost flat, and so in this last case the microcontroller turns off all the LEDs in the flashlight. The LEDs automatically light again when the supply voltage increases sufficiently.
If the battery supply is at 8 V then 1.5 V will appear on pin 3 of IC1 and D2 will glow green. If the supply falls to 6 V then the microcontroller will measure 1.14 V and D2 will glow orange. At 5.4 V, with a measured voltage of just 1.0 V, the LED glows red and the flashlight turns off.
When designing the circuit, care was taken to minimize its power consumption: it would be rather a pity to throw power away needlessly in a LED dimmer! It is for this reason that the battery level indicator LED flashes in all three of its states.
Schottky diode D1, type BAS70 or equivalent, protects input PB4 from possible excess voltage. Capacitor C1 (10 nF) serves to debounce switch S2. Should your switch be of a particularly indecisive nature you may find that this is not sufficient and hence that the dimming action is not smooth: before replacing the switch, try increasing the value of C1, say to 100 nF, and see if that solves the problem.
A type AP1117 voltage regulator provides power for the circuit at 3.3 V. Although the LED chain operates at a rather higher voltage (typically 7.2 V) the microcontroller is isolated from its supply by the MOSFET that switches its power.

Software

The program [1] was written using BASCOM-AVR. It is easy to modify the code, and only the demonstration version of the compiler is needed. Having set up variables with the thresholds for the various battery states, we initialize `Timer0`, which handles PWM signal generation, and `Timer1`, which is responsible for the flashing of the battery status indicator LED. The code spends most of its time waiting for a command from the user. Further down the code are the routines for indicating the charge status of the battery and for applying the thresholds to the A/D converter results to determine the state of the switch and pushbutton.
In 'flash' mode (when S1 is pressed) the LEDs are driven using the maximum possible pulse width. A brief time-out filters out any contact bounce that might lead to unexpected operation. The comments provided in the listing should help guide you if you decide to make any changes to it. For example, you may wish to increase the speed at which the brightness increases or decreases. This can be done by reducing the value of the variable `Pas` or by changing the PWM period from the value provided. The unit can be made to respond more quickly to S1 by reducing the debounce timeout in variable `Tempo_bp_on_value`. Finally, you might want to alter the initial PWM value (and hence brightness at switch-on) from the default (`Pwm = 100`).

Construction

The printed circuit board design is shown in **Figure 2**. The compact layout is achieved using surface-mount components, and this allows the board to be fitted easily into a cylindrical-style flashlight. However, we did encounter some unexpected problems with certain models where the transparent tube was so firmly glued to the other parts that it proved impossible to separate them without damage.
Once the flashlight is disassembled the only electrical modification to be made is to disconnect the common ground connection of the LED chains (which are often arranged as thirty series pairs all wired in parallel). The red cross in the circuit diagram (Figure 1) shows where the connection is broken. A wire must be soldered at this point connecting the bottom of the LED chain (or chains, as the case may be) to the terminal on K3 which in turn connects to T1. Two further wires connect the flashlight's ground connection and the positive battery terminal to K1. Then all that remains to be done is to make two holes in the flashlight's body where S1 and S2 can be mounted.

(140019)

Internet Link

[1] www.elektor-magazine.com/140019

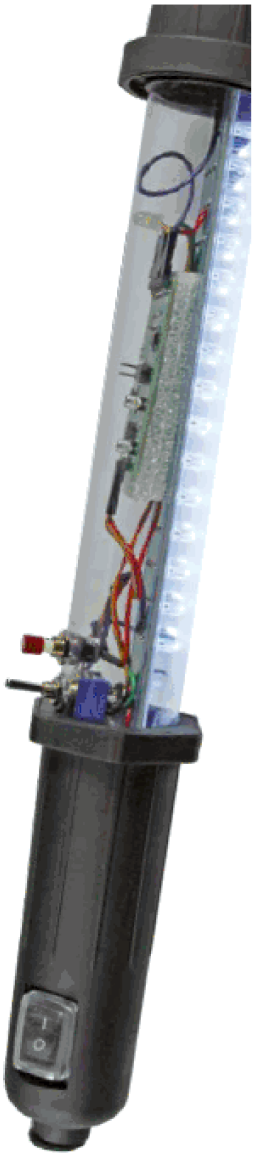
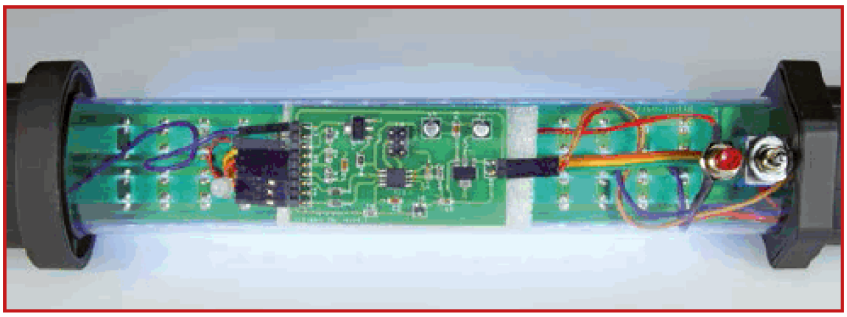


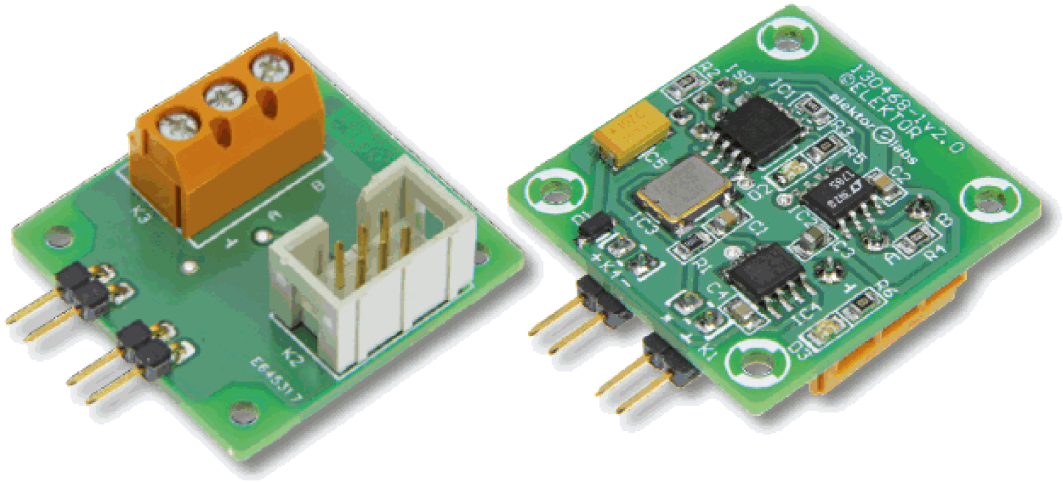
Figure 3.
If luck is on your side you will find a flashlight model like this one that can easily be disassembled and then reassembled with the dimmer circuit inside.



Temperature Sensor Board with RS-485 interface

Design:
André Goldberg,
Mauk van der Laan
and
Ton Giesberts

Text: **Jens Nickel**



Temperature sensors are needed in many automation applications, and the RS-485 interface allows reliable communication of data even over long distances. Our compact temperature sensor board is equipped with an ATtiny microcontroller and an RS-485 driver, and it is possible to connect several sensors in parallel to one board. In addition we present some example firmware which communicates temperature readings using the ElektorBus protocol, and software to display the results on a PC provides the finishing touch.

Our series of articles on the ElektorBus in 2011 generated a lot of interest among readers. We received hundreds of e-mails containing useful hints and tips or describing homebrew projects using the bus. One notable fan of the ElektorBus is André Goldberg, who built an ambient temperature measurement and control system. An important aspect of the project is the use of ElektorBus nodes with temperature sensors, which transmit readings to his PC.

Part of the appeal of the ElektorBus protocol is its simplicity. For example, each message is exactly 16 bytes long. The payload from a sensor board can carry four integer readings (in the range –1023 to +1023) to a central control unit: these might represent the outputs of four connected temperature sensors. A message going in the opposite direction can, for example, instruct the sensor unit whether to report temperatures in Celsius or Fahrenheit, or specify the interval between consecutive readings. The ElektorBus protocol provides commands for all of the above, see [1].

An RS-485 interface is used for communication. In half-duplex mode this requires two signal wires, and a separate ground must also be provided (see [2]). RS-485 is in theory highly immune to interference, and at the data rate used on the ElektorBus (9600 baud) communication over distances in excess of 30 m is possible.

Mini bus nodes

We have previously published circuit designs and printed circuit boards for ElektorBus nodes [3][4]. However, for many applications these boards are physically too large. André Goldberg gave some

thought to the question of how small the a bus node board could be made: the design that he delivered to our labs measures just 18 mm by 26 mm. It includes an ATtiny microcontroller in an SMD package, which offers six GPIO pins. The RS-485 driver is of course also an SMD device. Solder pads are provided to connect the bus lines, and likewise the in-system programming pins of the microcontroller and two GPIOs are also only brought out to solder pads, all in the interests of saving space. The external oscillator was also dispensed with, and the temperature sensor is a DS18S20 one-wire device. This consumes only one GPIO pin on the microcontroller as it uses a special asynchronous protocol for communication that removes the need for a clock signal to accompany the data signal. The data signal even provides power to the sensor: see the datasheet [5] for more details. Furthermore, several one-wire sensors can be connected to a single pin on the microcontroller: each sensor includes a preprogrammed unique 64-bit ID code, and the microcontroller can use the ID to address a particular sensor and receive a reading from it. The DS18S20 outputs 9-bit readings with a resolution of 0.5 °C.

Challenges ahead

The proposed hardware design creates several challenges on the software side.

- The ATtiny45 has no hardware UART that can be used to drive the RS-485 interface IC. This means that the UART (both transmit and receive parts) needs to be implemented in software. An advantage is that the two GPIO pins used can be chosen arbitrarily.

- A library is needed to read the one-wire sensor devices, using one further GPIO pin. Since we may have several sensors connected to the same pin, the microcontroller will also have to store the IDs of the individual devices so that the results can be output in the correct order.

- The ATtiny does not have enough program memory to contain the whole ElektorBus protocol library that we have described previously [6][7]. The required parts of the protocol will need to be reimplemented by hand.

André Goldberg decided to take advantage of two open source libraries, one for the software UART and one for the one-wire bus. During the configuration process the addresses of the individual sensor devices are read out and stored in the microcontroller's EEPROM. The readings from the four sensors are multiplied by ten so that the temperatures in Celsius with a resolution of 0.1 °C are represented as integers in accordance with

Component List

Resistors (0805)
R1 = 2.2kΩ
R2,R3 = 4.7kΩ
R4 = 120Ω
R5,R6 = 1kΩ

Capacitors
C1,C2,C4 = 100nF 25V, 10%, X7R, SMD 0805
C3 = 10µF 25V, 10%, X5R, SMD 0805
C5 = 100µF 16V, 20%, tantalum, SMD Case F

Semiconductors
D1 = PMEG2010AEH
D2 = LED, yellow, SMD 0805
D3 = LED, green, SMD 0805
IC1 = ATtiny85-20SU, SMD SO-8S2
IC2 = LT1785, SMD SO-8
IC3 = 8MHz quartz oscillator, 5x7mm, SMD (LF SPXO019079)
IC4 = 78L05

Miscellaneous
K1,K4 =2-pin pinheader, 0.1 inch pitch, right angled
K2 = 6-pin (2x3) pinheader or boxheader, 0.1 inch pitch
K3 = 3-way PCB screw terminal block, 0.2 inch pitch
PCB ref. 130468-1 v2.0

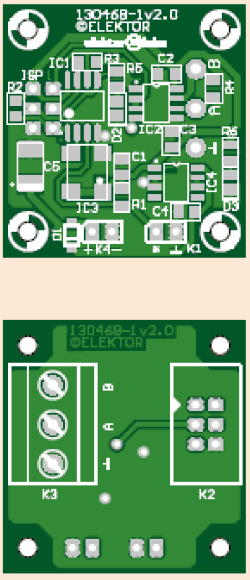


Figure 1.
The single-sided circuit board measures just 1.14 by 1.25 inch (29 mm by 32 mm). The headers are mounted on the underside.

the ElektorBus protocol. The remaining bytes in the ElektorBus protocol packet are 'manually' assembled into the code.

The circuit

In the Elektor Labs old hand Ton Giesberts immediately set about thinking how to 'Elektorize' the board. We decided to allow for the possibility of soldering in header pins or screw terminals for all the connections if desired, and we added a crystal oscillator for more reliable communication: this is particularly important if the circuit is to be subject to extreme variations in temperature. Nevertheless Ton managed to keep the board small: the final design measures about 31 mm by 32 mm (see **Figure 1**) [8]. The board is double-sided, with the ElektorBus screw terminals, the two-by-three programming header and the two two-pin headers for the power supply and for connecting the one-wire sensors located on the back of the board. The SMD components can be soldered by hand.

Our freelance colleague Mauk van der Laan found a significant improvement that could be made to the circuit. Instead of using two of the six GPIO pins for the transmit and receive connections for the software UART, we use only one pin, switching its function between input and output as needed. Because communication on the RS-485 bus is half duplex, a node never needs to speak and listen simultaneously; collisions between messages have to be avoided. The advantage of saving a GPIO pin is that we can use it to drive an LED to indicate the status of the node.

The circuit diagram is shown in **Figure 2**. The central component is the ATtiny85 [9], which has more flash memory than the ATtiny45. All six port pins are used, in some cases for more than one purpose. The one-wire temperature sensors are connected to PB4 via header K1. PB3 is driven by the SMD crystal oscillator module. The SPI interface on PB0, PB1 and PB2 and the reset pin PB5 form the ISP interface that allows code to be loaded into the AVR microcontroller. In normal operation PB0 drives the status LED.

The level on PB1 determines whether the RS-485 transceiver is in 'transmit' mode (PB1 high) or 'receive' mode (PB1 low). PB2 is the pin which, as described above, carries the UART data being transmitted or received.

The RS-485 signals are connected to K3. The 120-Ω termination resistor can be fitted or omitted as required.

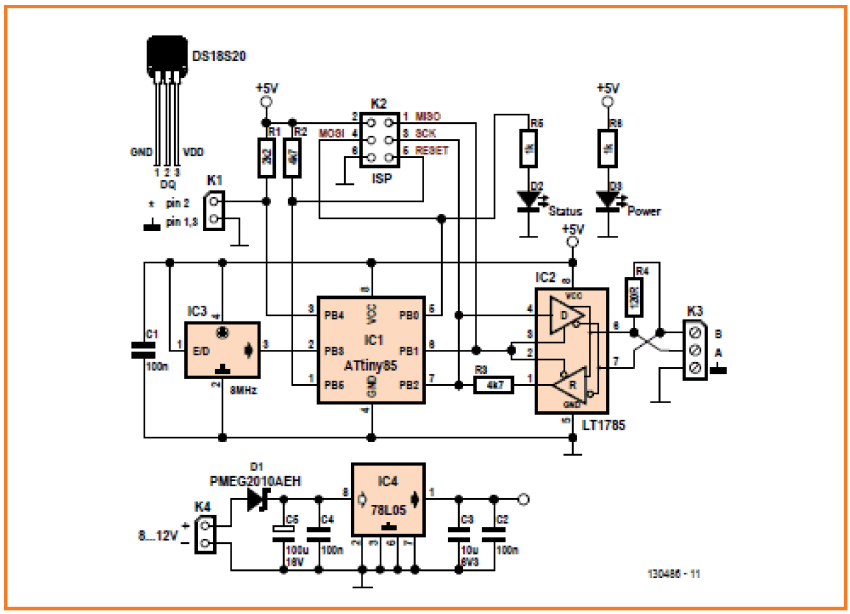


Figure 2.
The circuit centers around the ATtiny85. Only one pin (PB2) is needed by the software UART as transmission and reception never occur simultaneously.

Configuration

We provide standard firmware to run in the ATtiny85: it is of course open source and is available for free download at [8]. Mauk van der Laan has incorporated in the code a software UART library, a one-wire interface library (modified from an Arduino library) and a small ElektorBus interface. The whole thing is controlled by a kind of operating system that provides rudimentary multitasking (see the text box for more details). More advanced users will find studying the commented C++ code very rewarding.

Up to four DS18S20 temperature sensors can be connected in parallel to K1. In this configuration the datasheet instructs us to tie the VDD pin of each device to ground so that each sensor derives its power from the data line. The software uses an area of EEPROM ('slot one' to 'slot four') in the ATtiny to store the IDs of the temperature sensors. When power is applied to the node the microcontroller extracts the ID from each sensor over the one-wire bus. It then compares them with the stored IDs. If a stored ID is not found on the bus, then that ID is deleted and the storage slot becomes free. If an ID is seen on the bus that does not match a stored ID, it is stored in the first free storage slot.

It is now clear how we configure the system. First attach just one one-wire sensor and apply power to the board. The ID of this sensor will be stored

in the first storage slot. It is a good idea to label this first sensor '0' (assuming that programming in C has taught you to count from zero; if you are a BASIC programmer you may prefer to label it '1'!). Then disconnect the power supply, add the second sensor in parallel at K1 and power the board up again. This will store the ID of the second sensor in the second slot. Repeat the process for the third and fourth sensors.

The progress of the configuration process is neatly indicated by the status LED. When the software starts running the LED blinks rapidly. It then flashes four times, each flash being either short or long. A short flash corresponds to an unused storage slot, a long flash to a recognized ID address. If it is desired to remove a temperature sensor (even if its storage slot is not known) the system can simply be powered down, the sensor disconnected, and the system powered up again. The corresponding slot will be freed. The process described above can then be repeated to add a new sensor.

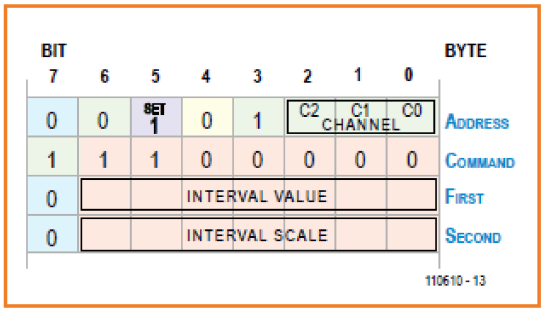


Figure 3.
Structure of a message to set the interval between readings on the temperature sensor board (only bytes six to nine shown). 'Interval scale' is 9, which is the code for tenths of a second. The 'channel' bits in principle could allow different intervals to be set for different temperature sensors, but they are not used in the standard firmware.

ElektorBus interface

The four temperature sensors correspond to four 'subnodes'. Our standard firmware periodically sends the four temperature readings packed into a single ElektorBus message, taking advantage of the 'channels' within the message payload. Each channel can carry a value from −1023 to +1023 [1], and each channel occupies two bytes within the message payload. The four channels occupy bytes 6 to 13 of the payload, counting from zero. The transmitter address in bytes 4

and 5 is fixed at '5' in our node. Bytes 2 and 3 contain the receiver address, fixed in our standard firmware at '10'. Bytes 14 and 15 (checksum/CRC) are not used.

The standard firmware is designed for one-to-one communication rather than for use with a number of sensor boards on a single bus. In so-called 'direct mode' collisions are avoided very simply by having a fixed pattern of transmissions. The sensor node sends the temperature readings at specified intervals to the 'master' (normally the control software running on a PC). If the master wants to send a command to the sensor node it waits until it receives a message containing temperature readings and then sends its message in the pause immediately thereafter. At the normal bus speed of 9600 baud a message repeat interval of under 100 ms is possible; however, the conversion time of the sensor devices is considerably longer than this (it can be as high as 750 ms) and so a message repeat interval of a few seconds is a better choice.

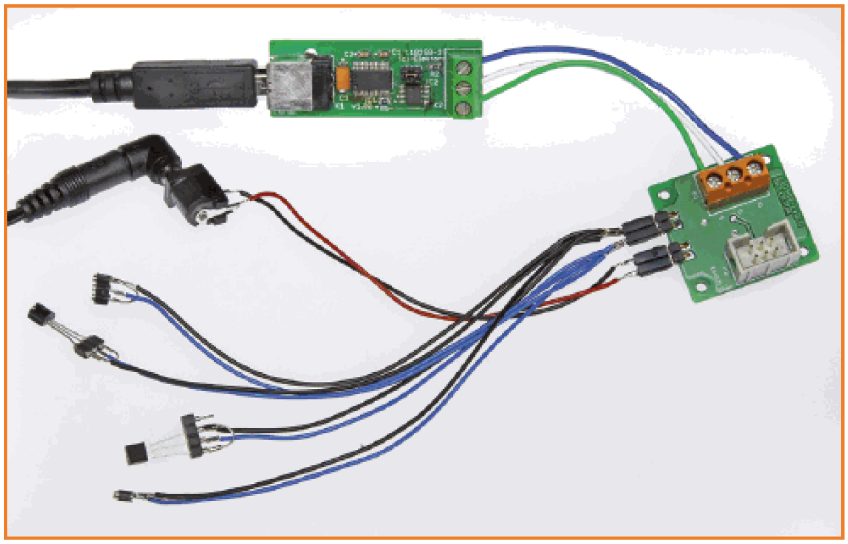


Figure 4.
The sensor board sends its readings to the PC over the RS-485 bus.

Mauk's firmware sends temperature readings (in Celsius, multiplied by ten) from all of the sensors it has identified at a default interval of 1 s. The node also listens for commands to change the interval, the other possibilities being 5 s and 500 ms (see **Figure 3**). The units used for the temperature values can also be changed to Fahrenheit and back to Celsius. When the sensor

Multitasking on the ATtiny

The standard firmware was written by Mauk van der Laan, a freelance software and electronics developer. Mauk's software is modular and includes a specially-designed operating system based on state machines to ensure the correct timing of the execution of tasks: see below.

The file 'Onewire.h' contains the one-wire interface library. 'OneWireTask.cpp/.h' is the state machine that is responsible for detecting the devices' IDs and for reading temperature values from them.
The software UART driver 'ElektorBusSw1w.cpp/.h' which is used for half-duplex RS-485 communication is derived from an open-source RS-232 library. The ElektorBus interface 'ElektorBus.cpp/.h' can, however, also work with a hardware UART. The line

```
#define ELEKTORBUS_DRIVER_INCLUDE
"ElektorBusSw1w.h"
```

in the file 'config.h' causes 'ElektorBus.cpp' in the standard firmware to access the software UART library. 'ElektorBusTask.cpp/.h' is the state machine that controls the bus interface.

The operating system offers so-called 'cooperative multitasking'. In contrast to preemptive multitasking a task is never interrupted by another: instead, it yields control voluntarily to the scheduler when it is ready. This means that no interlock mechanism is needed to prevent a task being paused at an inappropriate point. Also, unlike more heavyweight embedded operating systems such as FreeRTOS, the tasks do not have their own stacks. Each task just has a single state variable, and so memory usage is minimal: the operating system can run on an ATtiny with just 8 Kbyte of flash memory and 2 Kbyte of RAM.

The individual tasks are implemented using state machines. The entire code for each task is expressed in a switch-case block: depending on the value of the state variable a different section of the code will be executed. When its actions are complete, the task calls the method nextState() which returns control to the main task (also called the 'runtime'). Alternatively the task can call nextDelay() with an argument specifying a delay in milliseconds. This constitutes a request not to call the task again until the specified time has elapsed.

The scheduler in the runtime maintains a list of running tasks, each of which is derived from the 'Task' class. The

node receives a command message of this type, the next regular message it transmits will confirmation of its current units and scaling (and a message containing temperature readings will be lost). Subsequent messages will contain readings as normal. Since the reading in Fahrenheit can easily exceed 102, the sensor automatically switches from a scaling value of –1 (tenths of a degree) to 0 (whole degrees) when a command to use Fahrenheit units is received.

PC software

An ordinary terminal emulator program is sufficient for an initial test, at least to check that the board is indeed outputting readings at one-second intervals. We have also written some custom demonstration software for the PC. The user interface is as usual written in HTML and JavaScript. The software download [8] contains the software ElektorBusBrowser.exe and the folder UIBus, which should be dragged to your desktop.

The sensor board is connected using its RS-485 interface and the Elektor RS-485-to-USB converter [3], whose USB port is plugged into the PC (**Figure 4**). On launching the ElektorBusBrowser you must set the number of the COM port that is allocated to the RS-485-to-USB converter at the top of the screen, and then click on the 'Connect' button. If the sensor node is running and correctly configured the readings should now appear on the screen. Commands can also be sent to

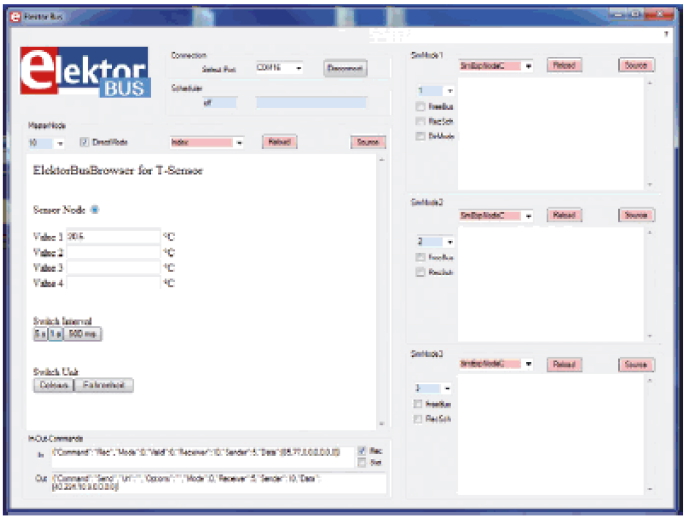


Figure 5.
The user interface on the PC is as usual based on HTML. Using the interface it is possible to adjust the measurement interval and the temperature units used.

scheduler calls the execute() method of each task in turn (unless it has requested a delay which has not yet expired). Each task should run for just a short time (perhaps a few hundred cycles) to maintain the illusion of full multitasking.

The following example code shows a task that flashes an LED.

```
class BlinkTask : public UserTask {

  enum States {Idle, BlinkingOn, BlinkingOff};

public:
  void start() {
    nextState(BlinkingOn);
  }
  void stop() {
    nextState(Idle);
  }

private:
  void turnOn();
  void turnOff();
  virtual void execute();
  virtual byte getTaskId() { return BLINKTASK_ID; }
};

void BlinkTask::execute() {
  switch(state) {
    case BlinkingOn:    // turn the LED on and wait
500 ms
      turnOn();
      nextDelay(BlinkingOff, 500);
      return;
    case BlinkingOff:   // turn the LED off and wait
500 ms
      turnOff();
      nextDelay(BlinkingOn, 500);
      return;
    default:     // invalid state: give up
      panic(1);
  }
}
```

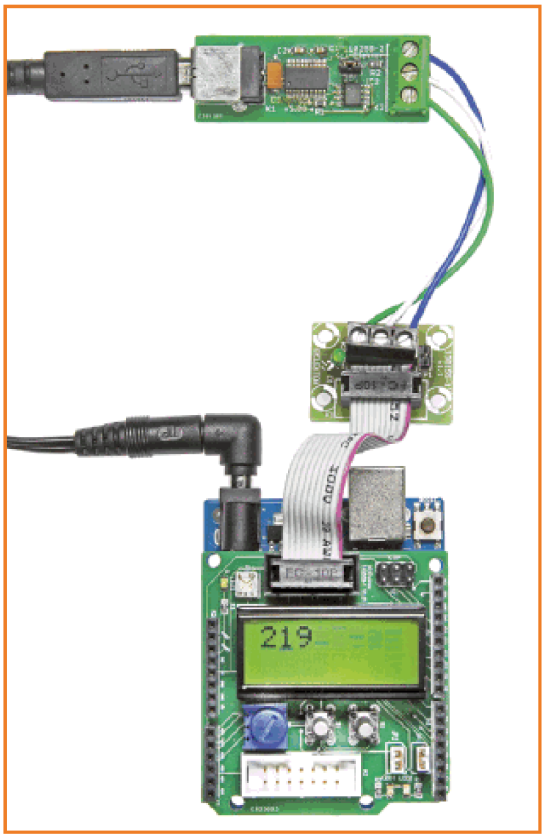


Figure 6.
Simulation using an Arduino Uno and the Elektor extension shield. This node behaves exactly like the temperature sensor board on the ElektorBus.

the node, but it is necessary to tick the 'Direct Mode' check box first.
The user interface is straightforward and should be self-explanatory (**Figure 5**). The labels '°C' and '°F' next to the temperature values only change when a message is received from the sensor node confirming the receipt of a command to change units. The change in scaling factor is also taken into account. To take a look at the HTML and JavaScript code, click on the 'Source' button.

If you also have the Andropod Android bridge board, you can couple the temperature sensor board with an Android smartphone or tablet [10].

The HTML and JavaScript user interface works equally well in the ElektorBusBrowserForAndropod, which can be downloaded for free from Google Play.

Simulation

Since the standard firmware for the board was not available at the time we were writing the PC control software, we simulated a temperature sensor node using an Arduino Uno board fitted with an Elektor extension shield [11] and an RS-485 module [12] (see **Figure 6**). The simulated temperature values are generated using a potentiometer and are shown on the display in tenths of a degree Celsius. The Arduino Uno sends the simulated value to the PC in channel 0, and of course this happens at fixed intervals. It also responds to control commands issued by the PC in exactly the same way as the standard firmware running on a real sensor node. The source code for the ATmega328 is also available for download [8]: it is based on the EFL [7] and uses the ElektorBus library. The 'Hardware' directory of the Atmel Studio project contains the code files for the Arduino Uno, the Elektor extension shield and the RS-485 ECC module. These make the higher layers of the software independent of the hardware and they are in turn independent of one another. Further software for the new shield will be presented in our next issue.

Of course the standard firmware and the ElektorBus protocol are not set in stone: you are free to write your own software and design your own protocols. Different sensors, switches and even actuators can be connected to K1: port pin PB4 can easily be used to generate digital signals or to measure voltages. If you do some up with a new use for the board, please drop us an e-mail to let us know or tell us at www.elektor-labs.com.

(130468)

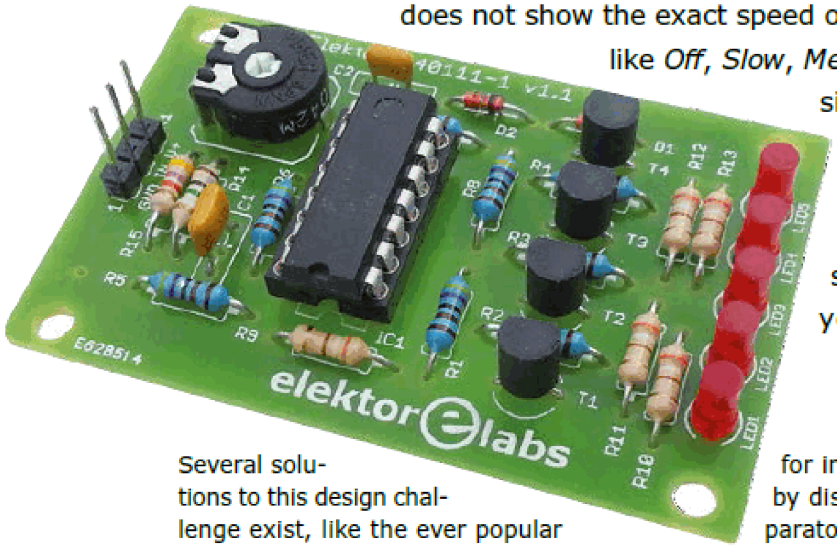
Web Links

[1] www.elektor-magazine.com/elektorbus
[2] www.elektor-magazine.com/110225
[3] www.elektor-magazine.com/110258
[4] www.elektor-magazine.com/110727
[5] http://datasheets.maximintegrated.com/en/ds/DS18S20.pdf
[6] www.elektor-magazine.com/120582
[7] www.elektor-magazine.com/120668
[8] www.elektor-magazine.com/130468
[9] www.atmel.com/images/atmel-2586-avr-8-bit-microcontroller-attiny25-attiny45-attiny85_datasheet.pdf
[10] www.elektor-magazine.com/110405
[11] www.elektor-magazine.com/140009
[12] www.elektor-magazine.com/130155

Dot Display Driver

By **Clemens Valens**
(Elektor Labs)

There are situations when it is useful to scale the dynamic range of a signal down to a few coarse sub ranges. As an example you can think of an indicator that does not show the exact speed of a motor, but only a few values like *Off*, *Slow*, *Medium*, *Fast* and *Too Fast*. Such signals can easily be obtained by averaging (i.e. low-pass filtering) amplitudes, frequencies or pulsewidths of signals produced somewhere in the system that you want to keep an eye on.



Several solutions to this design challenge exist, like the ever popular LM3914 (also available as NTE1549, and its siblings LM3915 and LM3916) with its convenient moving dot mode, or a microcontroller with an analog input. The LM3914 is easy to use but it does not offer much control over the way the input signal is quantized. A microcontroller gives you all the flexibility you can wish for, but it requires programming. The circuit presented here allows full control of how the input signal is quantized but it does not require any programming apart from calculating a few resistor values.

The circuit is straightforward. A multi-output voltage divider is dimensioned to cover all the ranges that matter. Each voltage divider output is compared to the input signal; when the latter exceeds the former, the corresponding comparator output swings High. This is all standard stuff, but not good enough, since when the input exceeds a certain level, all the comparators below this level will go High instead of just the last one. Our goal is to have only one active output for each level, not several. To paraphrase the LM3914's datasheet, our circuit must be in 'dot' mod, not 'bar graph' mode.

Adding (moving) dot mode can be achieved by making the higher level comparators control the comparators below them in some way, for instance by forcing their outputs Low or by disabling them. In this circuit every comparator controls the output of the comparator directly below it with a PNP transistor. So how does this work?

When two consecutive comparator outputs are Low, the base and emitter of the transistor between them will both see the same voltage. Consequently the transistor will switch off and so will be the LED driven by it. When the lower comparator output goes High while the upper stays Low, the transistor is switched on because its emitter voltage will be high enough with respect to its base voltage, hence the corresponding LED will light up. If now the upper comparator output becomes High too, the base and emitter voltages are again the same, causing the transistor to block and the LED go out. In short, an LED can only shine if the comparator output straight above it is Low while the comparator output straight below it is High. This assures that only one LED at a time can be on: the circuit is in dot mode.

There is a subtlety to this circuit that you have to be aware of. It lurks in the comparator output or, to be more precise, in its impedance. My experiments involved a single-supply rail-to-rail (SS-R2R) opamp type TS924, and everything worked perfectly fine. However, when I checked the availability of this opamp, I discovered that it has become obsolete and that its replacement,

the TS924A, is hard to get in a DIP package. So I switched to another SS-R2R op-amp, an LMC6464 that I happened to have handy. To my surprise, with this chip, my circuit no longer lit just one LED, but it also the LED right below it. Not too bright, but more than enough to be noticed. The cause is its rather high impedance making the output of the new op-amp drop as a quite steep function of the current it has to deliver. Suddenly the high level was not so high anymore, allowing the transistor driven by this output to turn on slightly because the transistor's base voltage no longer equaled its emitter voltage.

Checking the datasheets of both opamps explained it all. The TS924A, a pretty cool opamp that I heartily recommend, remains close to R2R with loads as low as 600 Ω whereas the LMC6464 is specified for loads of 25 kΩ or higher.

Okay, so the LMC6464 was a bad idea, but what about the good old LM324? With this chip the circuit worked almost fine even though the LEDs were a bit less bright, the LM324 high output level being about 1.5 V below its supply voltage. This created a problem for the lowest LED (*Off*) because I had tied the emitter of its controlling transistor to 5 V, too high for the opamp output to make the LED shine continuously. Adding two diodes in series with the emitter fixed this problem.
The input stage of the circuit is experimental; I just plugged some random component values into the schematic because it all depends on your input signal. I did my experiments with a 3-V 22-kHz PWM signal and got excellent results with C1 = 220 nF and R15 = 2.2 kΩ. The PCB was designed in such a way that capacitors of up to 10 µF should fit without problems. Higher-value parts should work too, but hovering over R15 and R14.
Adjust P1 to scale the quantization levels. Note that the scale in the schematic is linear thanks to resistors R5-R8 having identical values, but this is by no means obligatory.

Instead of the LEDs you can mount a pinheader to connect the board to, say, another one with relays on it. Do not forget to buffer the outputs if you have to drive heavy loads (to avoid the problems described above). Think ULN2003, for instance.

(140111)

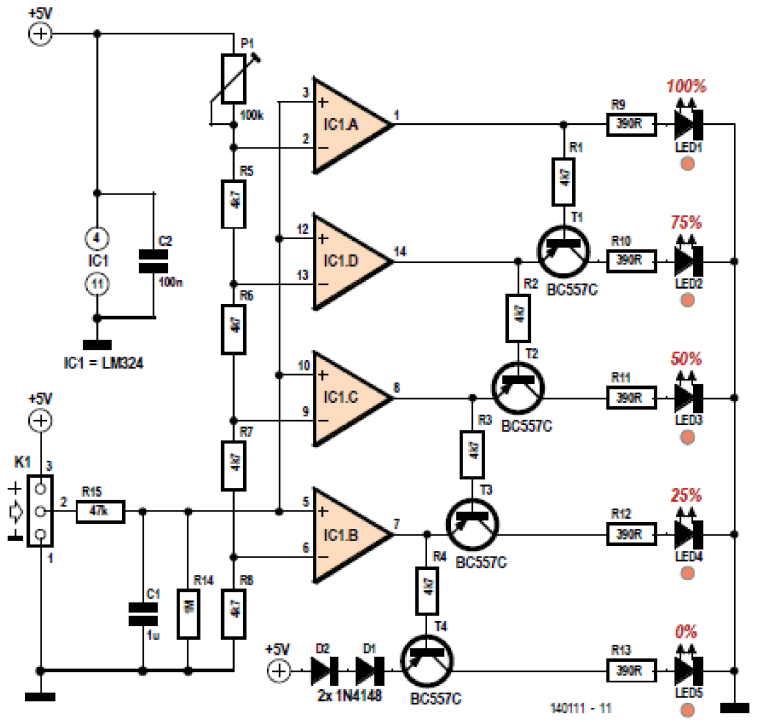


Figure 1. Dot Display Driver (DDD) schematic.

Web Link

http://www.elektor-labs.com/node/4013

Component List

Resistors
(0.25 W)
R9,R10,R11,R12,R13 = 390Ω
R1,R2,R3,R4,R5,R6,R7,R8 = 4.7kΩ
R15 = 47kΩ
R14 = 1MΩ
P1 = 100kΩ, trimmer

Capacitors
C2 = 100nF
C1 = 1µF

Semiconductors
D1,D2 = 1N4148
IC1 = LM324
LED1,LED2,LED3,LED4,LED5 = LED, red, 3mm
T1–T4 = BC557C

Miscellaneous
K1 = 3-pin pinheader, 0.1" pitch
PCB # 140111-1

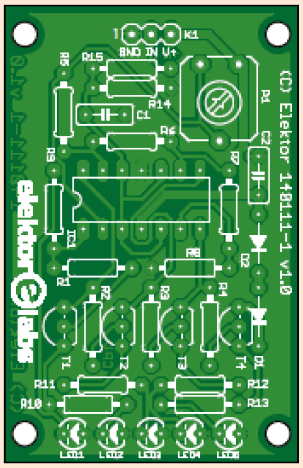
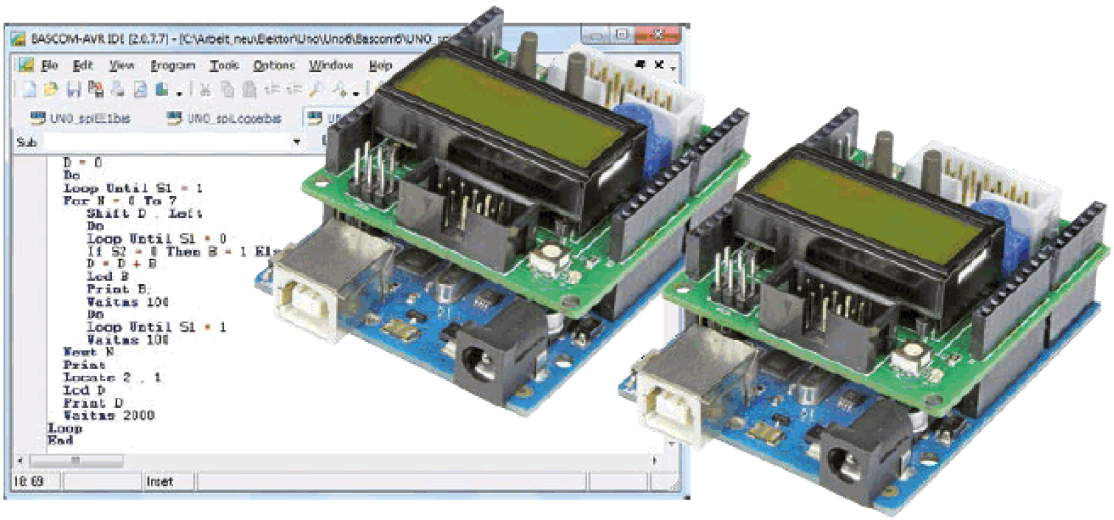


Figure 2.
For your convenience a small board was designed for the Dot Display Driver.

Microcontroller BootCamp (6)

The SPI interface



By
Burkhard Kainka
(Germany)

Serial communication with each unit of information following the previous one is actually the normal situation. That's how we talk to each other in person or by phone, read and write text—or in the case of Retronics: send telegrams.
In many cases all you need is a single line to transmit data. However, adding a clock line makes things more reliable. Let's find out.

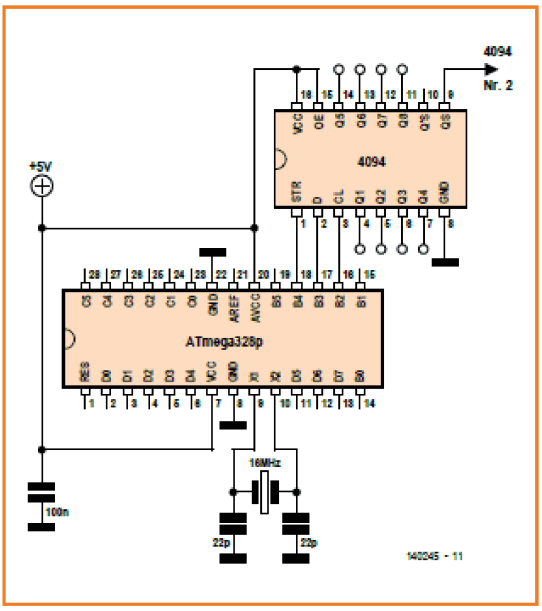


Figure 1.
Connecting a type 4094 shift register.

On the Serial Peripheral Interface (SPI) bus, the actual data travels bit by bit over one line—for example from a microcontroller to a display, an EEPROM or an SD card. In most cases it's also desirable to be able to read data, so you need a second line to provide a return channel for the data. There's yet another part to the picture: a clock line. The clock signal always clearly indicates when the next bit is available on the data line. That eliminates the need for precise agreement on data timing, which both parties have to supervise with timers. With these three lines, data transmission is fairly bombproof.

Port extension with a shift register

The first thing we want to try out is actually not an SPI interface, but instead something entirely different. Shift registers have been around for a long time (before microcontrollers were even

invented) and are good for understanding how serial data transfer works. They can also be put to good use in combination with a microcontroller. That's because port lines are always scarce, especially on Arduino boards. A port extension with a shift register can help ease the scarcity. With a type 4094 8-bit shift register, you need three lines to talk to it and you end up with eight new outputs. You can increase this to 16 by connecting a second shift register, or even 80 if you connect ten shift register ICs in series. If you need a lot of outputs, that's an especially low-cost way to meet the requirement.

Figure 1 shows the connections to the Uno board. The 8-bit shift register has a clock input (CL) and a data input (D). The data is applied to the D input one bit at a time, starting with the most significant bit, and a rising edge is applied to the clock input CL for each bit. The data is shifted through the individual flip-flops of the register step by step with each clock pulse. There is also the strobe input STR. When a pulse is applied to the strobe input, all the data present in the shift register is transferred to the type-D flip-flops connected to the output pins. You could also tie the strobe input to the supply voltage Vcc, but then all the intermediate results of each shift operation would appear on the outputs. By contrast, if you apply a strobe pulse after all the bits have been shifted in, you only see the final result on the outputs.

The software for all this (Listing 1) is simple. To output a byte D, the code first copies the most significant bit to the bit variable B (`B = D.7`) and puts it on the corresponding port pin. It then generates a positive clock pulse on CL with a length of 1 millisecond. A microsecond would also be sufficient, but the slower output is easier to see on an oscilloscope. After the clock pulse, a shift instruction (`Shift D , Left`) causes all bits of D to be shifted left by one position. This puts what used to be bit 6 on the output. This process is repeated until all eight bits have been shifted out. At the end comes the strobe pulse, and then all eight bits are present at the outputs of the 4094.
The code continuously increments the data byte to be transferred so the outputs of the shift register change while the program is running. The current value is shown on the LCD if the Elektor Extension shield is fitted, and it is output to the terminal emulator. This makes it easy to compare the output levels of the shift register to the digital value being output.
If you need more than eight outputs, you can use the Qs output of the 4094 IC. Each bit that is clocked into the shift register appears at the

Listing 1. Output using a shift register.

```
'-------------------------------------
'UNO_shift.BAS  Shift Register 4094
'-------------------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
$baud = 9600

Dim Dat As Byte
Dim D As Byte
Dim N As Byte
Dim B As Bit


Sr Alias Portb.4      '4094 pin 1
Da Alias Portb.3      '4094 pin 2
Cl Alias Portb.2      '4094 pin 3
Config Portb = Output

...

Dat = 0
Do
  Cls
  Lcd Dat
  Lcd "  "
  Print Dat
  D = Dat
  For N = 1 To 8
    B = D.7
    Da = B
    Waitms 1
    Cl = 1
    Waitms 1
    Cl = 0
    Waitms 1
    Shift D , Left
  Next N
  Sr = 1
  Waitms 1
  Sr = 0
  Waitms 1
  Dat = Dat + 1
  Waitms 500
Loop
End
```

Qs output eight clock pulses later. The D input of the next shift register can be connected to this output. In this way you can connect as many 4094 ICs in series as desired, with the clock and strobe lines connected to all of them in parallel. Of course, the software will have to be modified accordingly. First it shifts out all of the bits necessary to fill the chain of shift registers (e.g. 16 with two ICs or 80 with ten), and then it outputs the common strobe pulse.

Manual data transmission

Although you only need one line for the data, you also need a clock line when you use a clocked serial interface, as in the above example with a shift register or with the SPI bus. If you compare this with Morse telegraphy, for example, you can see the difference. There both parties have to agree on the transmission rate, and no breaks are allowed within an information unit. For example, if a radio operator wants to send an "X" (dash dot dot dash) and stops in the middle to scratch his head, the two characters "N" (dash dot) and "A" (dot dash) are sent instead. The situation is exactly the same with an asynchronous serial data interface, where both parties have to agree on the baud rate. After the transmission of a byte has started, all of the bits must be sent within a precise time frame. By contrast, with SPI the timing is not critical and any desired delays are allowed. The additional clock signal makes the transfer entirely independent of the speed. No matter whether the data rate is just one bit per minute or a million bits per second, the data will

Listing 2. SPI master and slave (very slow).

```
'-------------------------------------
'UNO_spi1.BAS  Shift in/out
'-------------------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
$baud = 9600

Dim D As Byte
Dim B As Bit
Dim N As Byte

S1 Alias Pinc.0
Portc.0 = 1
S2 Alias Pinc.1
Portc.1 = 1
Led1 Alias Portc.2
Ddrc.2 = 1
Led2 Alias Portb.2
Ddrb.2 = 1

...

Do
  D = Rnd(255)
  Cls
  Lcd D
  Print D
  Locate 2 , 1
  For N = 0 To 7
     B = D.7
     Lcd B
     Print B;
     Led2 = B
     Waitms 300
     Led1 = 1
     Waitms 200
     Led1 = 0
     Waitms 500
     Shift D , Left
  Next N
  Led1 = 0
  Led2 = 0
  Waitms 2000
  Cls
  Print
  D = 0
  Do
  Loop Until S1 = 1
  For N = 0 To 7
     Shift D , Left
     Do
     Loop Until S1 = 0
     If S2 = 0 Then B = 1 Else B = 0
     D = D + B
     Lcd B
     Print B;
     Waitms 100
     Do
     Loop Until S1 = 1
     Waitms 100
  Next N
  Print
  Locate 2 , 1
  Lcd D
  Print D
  Waitms 2000
Loop
End
```

be transferred properly. On an SPI bus there is always an SPI master and an SPI slave. Data can travel in both directions, but the master ways generates the clock signal.

You can try this for yourself manually, where you (as the user) assume the role of master. You can transmit a byte by pushing the two buttons S1 and S2 (**Figure 2**). This is not the usual way of doing things, but it helps you understand exactly how it works. One of the buttons is for the data, and the other is for the clock. Aside from that there's nothing new you have to learn, since you already know how a byte is put together. Here's how it works: First you send bit 7. If it is a '1', you press and hold button S2; otherwise you don't. Then you press button S1 briefly without changing the state of S2. The receiving end (the slave, which in this case is the Arduino board) then knows when it should read the bit from the data line. Now you repeat the process for bit 6, bit 5, and so on until bit 0.

The program in **Listing 2** displays the data in both directions. At first the microcontroller is the master and you are the slave. A byte with a random value is sent, with the clock signal indicated by LED1 and the data indicated by LED2. If you watch carefully, you can read the transmitted byte. However, that's not easy, so the byte is also shown on the LCD and sent to the terminal emulator. The individual bits also appear one after the other on the LCD and the terminal emulator screen:

```
83
01010011
```

Then the roles change. Now you are the master, and your job is to send exactly the same byte back to the microcontroller. Here you can see that the ability to send data at any desired speed is a big advantage, since you can take all the time you want to decide which bit value to send next. For example, suppose you want to send the decimal number 100. Bit 7 corresponds to decimal 128, which is more than 100, so it is '0'. Next comes bit 6 with a value of 64, so it's a '1', and you're left with 36 still to send. This means that bit 5 (32) is a '1', which leaves 4. The next two bits (bit 4 = 16 and bit 3 = 8) are '0', bit 3 (4) is a '1', and the last two bits are '0'. Now you have sent the binary number 01100100, with each bit

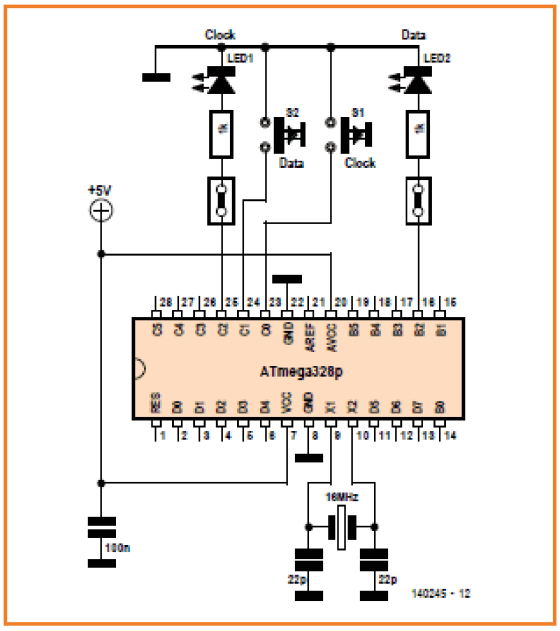


Figure 2.
Manual input and output.

marked by a clock pulse. It may not have been all that easy, but the microcontroller had no problem reading the data. You can regard this as a test of your concentration, and if the LCD shows the right result, you pass the test.

If you look closely at Listing 2, you will see that the bits are inverted when they are read. That's because pressing the data button yields a zero bit value. This is not especially intuitive, so the result is inverted when the bit is read to make things easier for you. Another interesting aspect is using the instruction `D = Rnd (255)` to generate a pseudo-random number. In fact, this always generates the same sequence of numbers, but the Bascom Help gives some suggestions for what you can do about this.

From microcontroller to microcontroller

In this example, data is sent over the SPI bus from one microcontroller to another. The data is this case consists of 10-bit readings from the A/D converter. This shows another advantage of SPI, which is that the data width is not fixed. No matter whether you send 8, 10, 12 or 16 bits, the procedure is always the same. If the only objective were to connect two microcontrollers together, it would actually be less effort to use an asynchronous serial interface with the TXD and RXD lines. The SPI bus, by contrast, is better for controlling and communicating with external

Listing 3. SPI master.

```
'------------------------------
'UNO_spi2.BAS  SPI Master
'------------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
$baud = 9600
Dim B As Bit
Dim Dout As Word
Dim N As Byte
Dim I As Byte

Sck Alias Portb.5
Ddrb.5 = 1
Mosi Alias Portb.3
Ddrb.3 = 1
Cs Alias Portb.2
Ddrb.2 = 1

Cs = 1
Mosi = 0
Sck = 0

Config Adc = Single , Prescaler = 32 ,
Reference = Avcc
Start Adc

Cls
Cursor Off

Waitms 200
Do
  Dout = Getadc(3)    'Pot
  Locate 1 , 1
  Lcd Dout
  Lcd "    "
  Cs = 0
  Waitms 20
  For N = 1 To 10
    Mosi = Dout.9
    Waitms 1
    Sck = 1
    Waitms 1
    Sck = 0
    Waitms 1
    Shift Dout , Left
  Next N
  Cs = 1
  Waitms 100
Loop
End
```

Listing 4. SPI slave.

```
'------------------------------
'UNO_spi3.BAS  SPI Slave
'------------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
$baud = 9600

Dim Addr As Byte
Dim B As Bit
Dim Dout As Word
Dim Din As Word
Dim N As Byte
Dim I As Byte


S1 Alias Pinc.0
Portc.0 = 1
S2 Alias Pinc.1
Portc.1 = 1
Sck Alias Pinb.5
Portb.5 = 1
Mosi Alias Pinb.3
Portb.3 = 1
Cs Alias Pinb.2
Portb.2 = 1
...

Do
  Do
  Loop Until Cs = 0
  Din = 0
  For N = 1 To 10
    Shift Din , Left
    Do
    Loop Until Sck = 1
    Din = Din + Mosi
    Do
    Loop Until Sck = 0
  Next N
  Do
  Loop Until Cs = 1
  Locate 1 , 1
  Lcd Din
  Lcd "    "
  Print Din
Loop
End
```

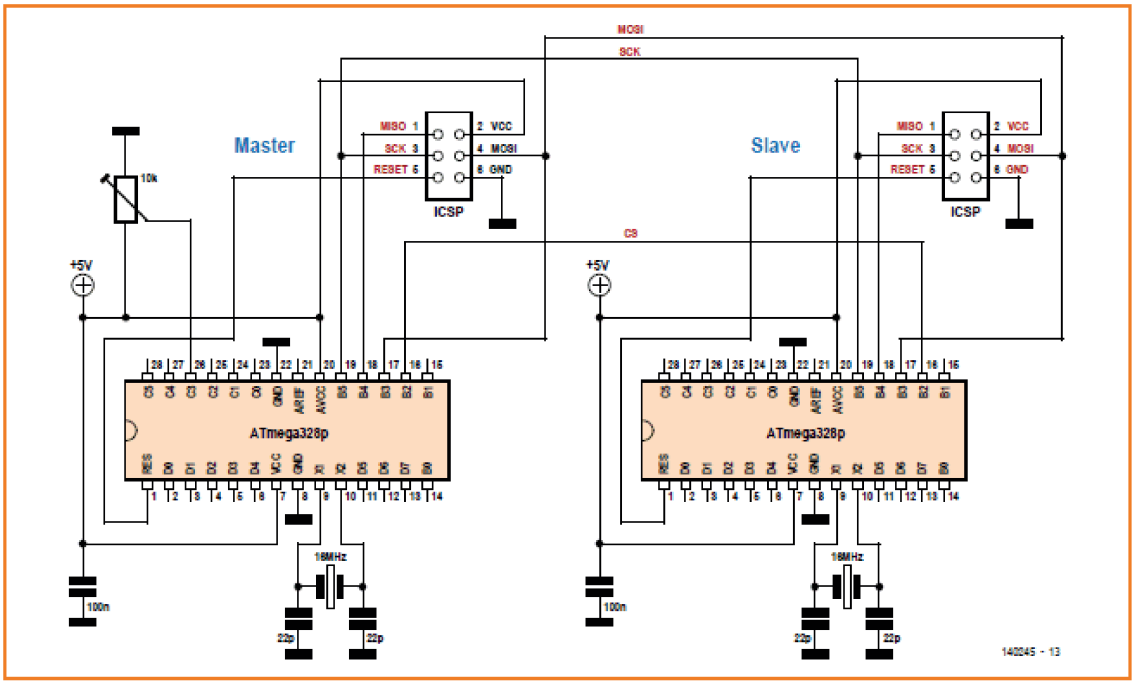


Figure 3.
An SPI connection between two microcontrollers.

hardware. Here the main purpose of the exercise is to illustrate the transmission protocol.
As previously with the 4094 shift register, a third line is involved here—in this case the chip select line /CS. The slash (/) means that the signal on this line is active Low. The chip select line allows you to connect several slave devices to a single master. In that case they share the data and clock lines, but each one has its own chip select line. When that line is low, the corresponding slave knows that it is selected. There's also another benefit from using a chip select line. If there is any delay in enabling the slave, there may be some confusion about which bits have already been transferred. However, if the slave waits until it sees a falling edge on its CS input (high to low signal transition), it knows that the transfer is starting. And if a noise pulse is read as a clock signal, the rest of the data for that transfer is trash, but on the next access everything is again as it should be.

The ATmega328 also uses the SPI bus for program download from an external programming device. The following lines are therefore available on the six-pin programming connector on the Arduino board and on the Elektor Extension shield (ICSP in **Figure 3**): the clock line Serial Clock (SCK) on B5, the write data line Master Out Slave In (MOSI) on B3, and the read data line Master In Slave Out (MISO) on B4. There is no chip select line, but the Reset line has the same effect because programming takes place with the Reset line pulled low. Now we want to use these lines exactly as intended. This has the advantage that we can use the hardware SPI unit of the microcontroller, if it has one. With hardware SPI we do not have use program code to put each bit individually on the data line as in the previous examples, and everything is a lot faster. However, we still need a chip select line, and in this case we use the B2 line for this purpose.
The master uses the MOSI line as the output and generates the clock and chip select signals (**Listing 3**). The process is slowed down a bit by three 1-millisecond delays so that all the signals can easily be seen on the oscilloscope. Besides, we don't want to make things too difficult for the slave. If you wish, you can test the boundaries by reducing the delays until transmission errors start to occur.
The three lines are inputs for the slave device (**Listing 4**). It constantly waits for specific signal edges on the /CS and SCK lines and then reads in a bit from the MOSI line. Since everything is handled by software here, the code must wait for each edge in a Do loop. This takes a bit of time, so data transmission must be slower than with a hardware SPI implementation. The received data is shown on the display and on the terminal emulator. When you turn the potentiometer on the master board, the change is visible on the slave.

SPI EEPROM 25LC512

There is a wide range of ICs available with an SPI interface, including A/D converters, memory devices and display drivers. Serial EEPROMs from Microchip and other companies are available at low cost and are widely used. The 25LC512 (not to be confused with the 24C512, which has a I²C bus interface) has a capacity of 64 KB, and it is a good solution when the 1-kilobyte capacity of the ATmega328's internal EEPROM is not sufficient. I²C EEPROMs are more widely used in the hobby realm, but SPI types are generally preferred in the professional realm because they offer especially high operational reliability.

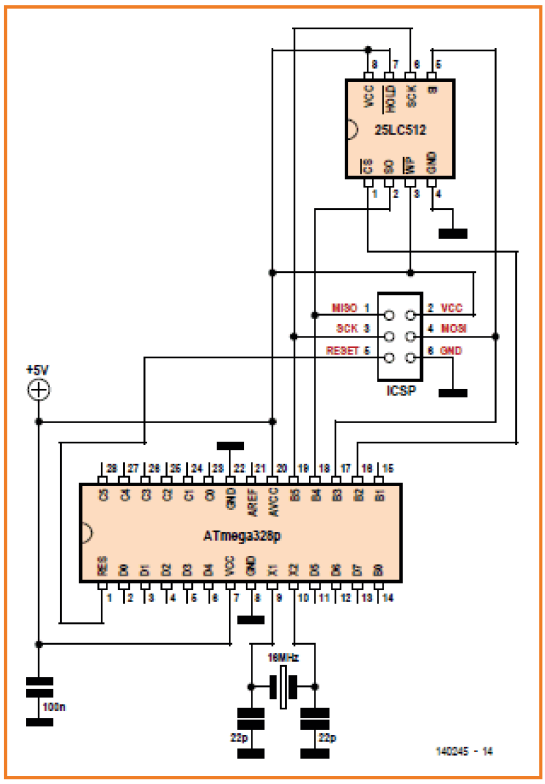


Figure 4.
Connecting a serial EEPROM.

Figure 4 shows the connections to the Uno board. The pins of the original SPI interface (2x3 pin header) are again used here. That makes it easy to build a convenient plug-in memory module by fitting the IC in a socket soldered to a small 6-pin socket header. You only have to connect one additional line. This is the chip select line, which is again assigned to B2 because the Reset line present on the connector cannot be used for this purpose.

Here there are two data lines. MOSI (Master Out Slave In) is the data output line and is connected to the Serial Input (SI) pin of the EEPROM, while MISO (Master In Slave Out) is used to read data from the Serial Output (SO) pin. The microcontroller is always the master, and it generates the clock on the SCK line. To go with this exercise, we have written a subroutine (subroutines are called "Sub" in Bascom; see the **inset**) that transfers data in both directions in a single go (**Listing 5**). Before the subroutine is called, the data to be sent must be placed in the global variable **Dout**, and after the call the received data is located in the variable **Din**. Of course there are situations in which data is only written, but in that case zero bits are usually sent in the other direction. The relatively complex data sheet for the 25LC512 tells you what has to be sent to the device, as well as when and how. After the chip select line has been pulled low, the memory chip first receives a simple byte command that specifies what action is to be performed. To read data from the memory, you have to send a '3' command followed by two address bytes forming the high-byte and low-byte portions of the address. After that as many data bytes as desired can be read out with automatic address incrementing (see **Listing 6**). The program displays the sequential addresses and the data bytes that are read out. A brand-new or fully erased EEPROM always delivers only the value 255. Now let's try to program some data. For that we use the byte command '2'. However, a bit of preparation is necessary first. Writing must be enabled by sending the command '6' (**Listing 7**). To check whether writing is enabled you can read the EEPROM status register, which requires sending the command '5'. Each action is only effective if you pull /CS low at the start and then return it to the high level at the end.

Listing 5. Reading and writing data over MOSI and MISO.

```
Sub Spioutin
  Din = 0
  For N = 0 To 7
    Shift Din , Left
    Din = Din + Miso
    If Dout.7 = 1 Then Mosi = 1 Else Mosi = 0
    Waitus 3
    Sck = 1
    Waitus 2
    Sck = 0
    Waitus 2
    Shift Dout , Left
  Next N
End Sub
```

Listing 6. EEPROM readout (excerpt).

```
Cs = 0
Dout = 3          'read
Spioutin
Dout = 0          'A8...A15
Spioutin
Dout = 0          'A0...A7
Spioutin
I = 0
Do
  Locate 1 , 1
  Lcd I
  Print I;
  Print "  ";
  I = I + 1
  Spioutin
  Locate 2 , 1
  Lcd Din
  Print Din
  Waitms 200
Loop
```

If the value in the status register is 2, the IC is enabled for write operations. Complicated? Yes, but that makes it especially foolproof.
Now we are allowed to write data to the memory, but even then there's something we have to consider. The memory space is divided into pages, which in the case of the 25LC512 have a size of 128 bytes. Each transfer is limited to a maximum of 128 bytes of data, or as many bytes as it takes to reach the next page boundary. After this you switch /CS high and then give the EEPROM enough time to actually store the data. According to the data sheet, 5 ms is enough

Listing 7. Enabling write access (excerpt).

```
Cs = 0
Dout = 6          'write enable
Spioutin
Cs = 1
Waitms 20
Cs = 0
Dout = 5          'read status
Spioutin
Spioutin
Cs = 1
Locate 1 , 1      'status
Lcd Din
Print Din
```

Subroutines

Subroutines are called "Sub" in Bascom, and they are used to allow a block of code to be called repeatedly from different locations in a program. To make this possible, each subroutine must be declared at the start of the program. This way the name of the subroutine is known to the compiler, so the it can be called using this name in the same way as Bascom functions.

```
Declare Sub Spioutin()
…
Dout = 6
Spioutin
…


Sub Spioutin
  Din = 0
…
  Shift Dout , Left
End Sub
```

You always have to be very careful with the variables used by a subroutine. In the case of the **Spioutin** subroutine, all of the variables it uses are "global" variables, which are dimensioned at the start of the main routine and are therefore valid in the entire program. However, you could do things differently and transfer the data to the subroutine when it is called:

```
Declare Sub Spioutin (Byteout as Byte)
```

In that case the variable **Byteout** would not be valid globally, but only within the subroutine. The subroutine call would then take the form:

```
Spioutin 6
```

This saves one line compared to the previously shown call.
You can also transfer a group of several variables to a subroutine in a subroutine call, as can be seen from the example of the Bascom **Spiout** subroutine:

```
Spiout Dout , 1
```

For novice programmers, it's generally safer to use only global variables in your own subroutines. For advanced programmers, on the other hand, selecting the optimal form of data transfer is a major consideration.

Listing 8. Using the software SPI (excerpt).

```
Config Spi = Soft , Din = Pinb.4 , Dout = Portb.3 ,
Ss = None , Clock = Portb.5 , Mode = 0
Spiinit

Cs Alias Portb.2
Ddrb.2 = 1
Cs = 1

     Cs = 0
     Dout = 4          'write disable
     Spiout Dout , 1
     Cs = 1
     Waitms 1
     Cs = 0
     Dout = 3          'read
     Spiout Dout , 1
     Dout = 0          'A8...A15
     Spiout Dout , 1
     Dout = 0          'A0...A7
     Spiout Dout , 1
     I = 0
     Do
       Locate 1 , 1
       Lcd I
       Lcd "  "
       Spiin Din , 1
       Locate 2 , 1
       Lcd Din
       Lcd "  "
       Print I ;
       Print "  ";
       Print Din
       Waitms 200
       I = I + 1
       If I >= 65535 Then Exit Do
     Loop
```

time for storing the data. If you exceed the page boundary (I tried it), the result is chaos. Then the data you find in memory is totally different from what you wanted to write to memory. For this reason, the example program carefully obeys the rules and writes 128 bytes to the first page from address 0 to address 127, with the data values in ascending order from 0 to 127. When the data is read back, that's exactly what you see. As usual, the entire program code (**UNO_spiEE1.bas**) can be downloaded from the Elektor website [1]. It performs the following actions in sequence:

- Command 6, write enable
- Command 5, read status register; display for 1 second

Listing 9. Data storage in the timer interrupt routine (excerpt).

```
Tim0_isr:
  '4000 µs
  Timer0 = 6
  Ticks = Ticks + 1
  If Ticks = 250 Then
    Ticks = 0
    U = Getadc(4)
    U = U / 4
    Addr = Seconds
    Addr = Addr And 127
    If Addr = 0 Then          'start of page
      Cs = 0
      Dout = 6                'write enable
      Spiout Dout , 1
      Cs = 1
      Waitus 100
      Cs = 0
      Dout = 2                'write
      Spiout Dout , 1
      Waitus 100
      Cs = 0
      Dout = 3
      Addr = High(seconds)
      Dout = Addr             'A8...A15
      Spiout Dout , 1
      Addr = Low(seconds)
      Dout = Addr             'A0...A7
      Spiout Dout , 1
    End If
    Dout = U
    Spiout Dout , 1
    Addr = Seconds
    Addr = Addr And 127
    If Addr = 127 Then        'End of page
      Cs = 1
    End If
    Print Seconds
    Seconds = Seconds + 1
    If Seconds = 0 Then Seconds = 65535
  End If
Return
```

IoT & the Search for a Protocol

By Jens Nickel

The 'Internet of Things' (IoT) is set to change our lives. In order for the diverse equipment to understand each other a consistent, application orientated protocol is needed. For example it's necessary to define a general purpose representation of measurement values from sensors and positional information for actuators. Ideally the protocol will not present too much of an overhead for a small, low cost microcontroller. At the same time it needs to be versatile enough to cater for the majority of applications that you might find, for example in a Smart Home environment [1]. This won't be a cakewalk, that's why in our March 2014 edition we appealed to the Community to help us out and send their ideas to our competition web page at www.iot-contest.com. In the mean time we have already received many interesting suggestions that got us thinking, as well as pointers to suitable existing protocols [2][3]. Now we would like to provoke some discussion by outlining a framework for a new protocol that is still 'work in progress'. This is your chance to make a difference—if you think the ideas are good then say so, if you can think of a better solution we want to hear it. This is going to be a collaborative effort and we think, with your help we will arrive at an optimal solution. Get involved and go to www.iot-contest.com!

(140298)

[1] www.iot-contest.com/index.php?content=infos

[2] www.xapautomation.org

[3] http://ansari-electronics.com/comlinksrv/

Suggested data format

Messages are passed between devices, for example as HTTP-POST data (this needs further discussion). Each message consists of a header together with blocks of information. The type of information block is defined by two key words (these are described as 'Case' or 'Mode', see below).

A few examples:
'Set Value': Sets an actuator to a value
'Current Value': A measurement value is transferred
'Query Value': Request a measurement value to be transferred (Polling)
'Set LimitMin', **'Set LimitMax'**: Set end limits to an intelligent sensor
'Set Interval': Set up sensor to send measurement at intervals
'Option Min', **'Option Max'**: The sensor indicates its range
'Option Value' (followed by physical units and size): The sensor indicates what it is measuring and the measurement units.

An outline of the message structure (optional information is bracketed), the // symbol means 'or'):

Message: Header + Block (+ Block (+ Block ...))
Block: Case + Mode + (Subnode +) (Location +) (Time +) Data
Case: 'Set' // 'Current' // 'Query' // 'Option'
Mode: 'Value' // 'Min' // 'Max' // 'Interval' // 'LimitMin' // 'LimitMax'
Subnode: *Integer*
Location: LocationString: *String*
Time: TimeString: *String*
Data: Value (+ Value (+ Value + ...))
Value: Number + (Accuracy +) Unit + Quantity
Number: *Float // Integer // Binary*
Accuracy: *Float // Integer*
Unit: UnitKey: *String*
Quantity: QuantityKey: *String*

In the end all elements of data types *Float*, *Integer*, *Binary* und *String* will be returned. Exactly how the elements (in ASCII format) are represented must be defined. Amongst other things we also need suggestions for the representation of location and time (LocationString, TimeString).

A versatile protocol will allow the messages to be made simpler or more complex. For example where Presets (predefined actuator positions) are not required or where sensor alarm information such as ('limit exceeded') must be accommodated in the message.

- Command 2, write 128 bytes starting at address 0
- Command 3, read memory starting at address 0; endless loop

Data logger

One practical application for the 64-KB memory is a data logger. The objective here is to acquire measurement data from the ADC4 analog input once per second and store the data. The memory will be full in approximately 18 hours.

You don't always have to program everything yourself, since Bascom has a lot of ready-made functions for many situations. In this case you have the option of configuring an SPI interface as a software interface using any desired port pins or as a hardware interface using the microcontroller pins designated for this purpose. The hardware SPI is especially fast and is commonly used for tasks such as driving graphical displays. However, this involves a whole lot of parameters that must be configured for each specific application, which requires a detailed study of the ATmega328 data sheet. Things are a bit easier with the software SPI function, and it provides a reasonably high transmission rate. Although writing your own SPI procedure from the ground up is not a bad idea because it allows you to implement the timing diagrams in the data sheets very clearly, the ready-made software SPI is more convenient and faster, which is why we use it here.
The interface configuration specifies which lines are to be used. For Din, Dout and Clock we use the familiar MISO, MOSI and SCK lines, which are already available on the ICSP connector. The SS line corresponds to the /CS line. In this case this line should not be operated automatically by Bascom because it is usually necessary to transfer a lot of data during a single active chip select phase. Consequently, this line (on port pin B2) will still be operated "manually". The `Mode = 0` setting is also important, because there are four different SPI modes.
The program excerpt in **Listing 8** shows how the software SPI is used to read data from the serial EEPROM. The instruction `Spiout Dout , 1` sends exactly one byte, which is transferred in the variable `Dout`. In the other direction, the instruction `Spiin Din , 1` reads one byte, which is then available in the variable `Din`. The entire program reads all the data from the EEPROM and shows the contents on the display and on the terminal emulator screen.
As usual, the entire program code (`UNO_spiLogger.bas`) can be downloaded from the Elektor website [1]. It is too large to be listed fully here. Pressing S1 starts a measurement run. It can be stopped at any time by pressing S2, after which the stored data can be read out.

A timer interrupt routine (excerpt in **Listing 9**) is used to control the timing during data acquisition. The voltage on ADC4 is measured and stored once per second. The 128-byte block size of the EEPROM is taken into account. At the start of each block, a write access is started and the current address is transferred, followed by 128 bytes of data. At the end of the block, the /CS line is pulled high to allow the EEPROM to store the entire block. Since the /CS line is connected to port pin PB2, LED2 on the shield is lit when the line is high. The LED therefore flashes each time a block of data has been transferred to memory.

(140245-I)

Web Link

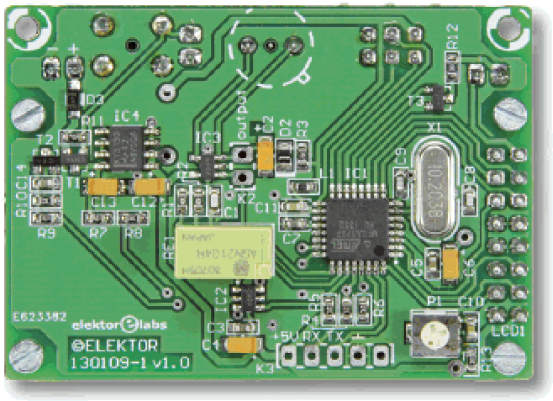
[1] www.elektor-magazine.com/140245

Tip for using the Arduino programmer in Bascom

Many readers who are using the original Arduino boot loader have encountered the following problem: if a program performs serial output, the Arduino programmer in Bascom does not work properly the next time and hangs. Some readers have discovered that this problem can be resolved by using the Arduino IDE before using the programmer again. Simply transferring the Blink program, for example, is sufficient.
However, there's an easier solution. After you launch the programmer in Bascom, briefly press the Reset button on the Uno board three times (or more) at roughly 1-second intervals. After this the programming function will work again, even when the previous program included a Print output. This has been discussed intensively in various topics on the Elektor forum (now at forum.elektor.com).
If you use the MCS boot loader on the Uno board, this problem does not occur. Another alternative is to use an external programmer.

Lux Meter

With 1 lx to 100 klx measuring ranges



By
Karl-Anton Dichtel
(Germany)

Conventional incandescent lamps have been phased out in many areas in the world, including the EU and Canada. What's left is halogen lamps, energy-efficient lamps and LED lamps, which in fact have better efficiency than incandescent lamps. However, if you want to know how much better they are, you basically have to trust the manufacturer's data. Although trust is good, measuring it yourself is better. The lux meter described here can help.

Particularly with low-cost LED lamps from low-wage countries, some of which are a lot cheaper than brand-name products, you can hardly be blamed for being a bit sceptical. For example, among the 500,000 items available on a well-known auction site there are exactly five 3-watt spot lamps with an asking price of 1 euro. The claimed light output of these lamps is 210 lumen. That sounds realistic, but is it actually true?

To avoid being forced to rely on such claims, you need an instrument that can reliably measure brightness. Such an instrument is useful for more than just checking whether a lamp works. For example, it's good for checking whether a particular workstation is adequately lit—and in many countries there are regulations governing this.
A lux meter also offers other benefits with lamps. For example, for a variety of reasons halogen lamps are popular for certain purposes. The brightness of halogen lamps tends to decrease over time, in part due to the accumulation of metal vapor deposits on the glass envelope. As a result, at some point you are getting more heat than light from the lamp. Even energy-efficient lamps get darker with age. Measurements can help you decide when it's time to replace a lamp that is no longer up to snuff.

Measuring light

The key element of every instrument is a sensor that converts the desired physical quantity into an easily measured voltage, preferably with a reasonably linear conversion characteristic. Of course, nowadays a fair amount of electronics is integrated into many sensors to enable them to output digital data directly. However, the traditional sensor for light is a photodiode. Along with types intended for the transmission of infrared signals, there are special, more accurate types for measurement purposes. The BPW21R used here is an example of the latter type, and it is suitably housed in a sturdy metal TO5 package with a glass window (**Figure 1**).

This hermetically sealed photodiode has a relatively large active area of 7.5 mm², which gives it high light sensitivity. This sensor is also very linear, as can be seen from the data sheet [1]. **Figure 2** shows the incredibly linear relationship between the short-circuit current and the illuminance. The log-log scale is necessary here because the sensor covers an extremely wide brightness range of more than seven decades. Even though this is a conventional discrete component, its specs are nothing to be ashamed of, which makes it the ideal light sensor for use in a DIY lux meter.

A photodiode generates a current that is proportional to the illuminance in lux, rather than the luminous flux in lumens typically used in lamp specifications. As 1 lux is equal to 1 lumen per square metre, it is theoretically easy to convert between the two units, but the area illuminated by a lamp depends on the distance and on the beam angle. Thanks to the Internet, you don't need a pencil and paper for this and you don't have to solve any algebraic equations. Various websites (e.g. [2]) have ready-made calculators where you can simply enter a value in lux or lumens and convert it into the opposite unit.

Measurement method

As already mentioned, the output from the photodiode is its short-circuit current. A highly linear relationship between illuminance and signal level can only be obtained by operating the photodiode under virtually short-circuit conditions, which means that the circuit for measuring the current must have a very low input impedance. However, the quick and dirty approach of connected a 1-ohm resistor across the sensor and measuring the resulting voltage drop is not very effective because the voltage at 1 lux would only be about 9 nV. Although microcontrollers with integrated A/D converters are usually the preferred choice for measurement tasks, the A/D converters of commonly used microcontrollers are not suitable for such low voltages. With the 10-bit resolution of a typical AVR microcontroller and a reference voltage of 1 V, the resolution is around 1 mV, which is five orders of magnitude greater than the full-scale value of the lowest measuring range. The idea of using an intermediate voltage amplifier to boost the voltage by a factor of 100 million (to obtain roughly 1 V at 1 lx) is unrealistic, since the signal would be completely buried in the noise (among other problems). We therefore need a different approach. The circuit in **Figure 3** shows how it can be done. The sensor D1 is connected across the inputs of an opamp (IC3), which has its non-inverting input connected to a reference voltage of approximately 0.65 V at the junction of R3 and D2. When light shines on D1, it wants to supply a current, which would cause the voltage on the inverting input of IC3 to drop. This is countered by negative feedback through R1 or R2. Consequently, the voltage at the output of the opamp rises to the level necessary to cause the current flowing through the feedback resistor to offset the current from the diode, so the voltage between the inputs of IC remains at zero. As a result, D1 is virtually short-circuited and therefore operates with a very linear characteristic. To obtain the highest possible measurement accuracy, we chose a Microchip MCP6061T for IC3 instead of a standard opamp. With extremely low input currents in the picoamp range and an offset voltage of only 150 µV, this opamp will not degrade the measurements. The converted sensor signal from IC3 is routed to connector K2 for test purposes.

Features

- Lux meter with 6 measuring ranges: 1 lx, 10 lx, 100 lx, 1 klx, 10 klx and 100 klx
- Measurement resolution in 1 lx range: 0.01 lx
- Autoranging
- Automatic power down after 1 minute
- Measurement data output over serial interface
- All components readily available
- Low power consumption
- Can be calibrated over the serial interface using a terminal emulator

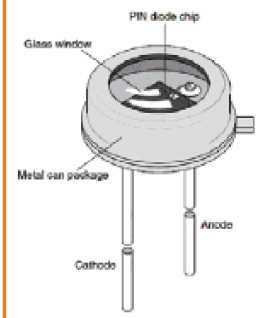


Figure 1.
Schematic depiction of the BPW21R package.

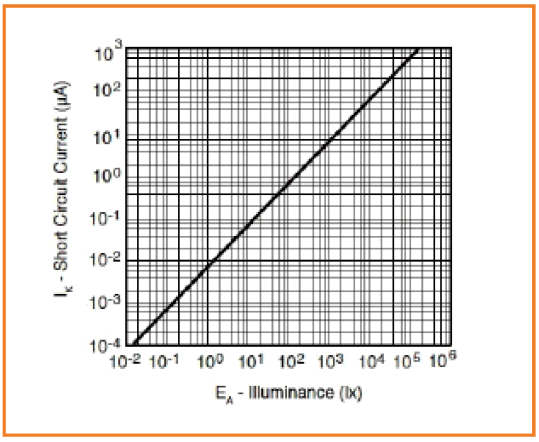


Figure 2.
The short-circuit current (I_K) versus illuminance (E_A) characteristic of the BPW21R.

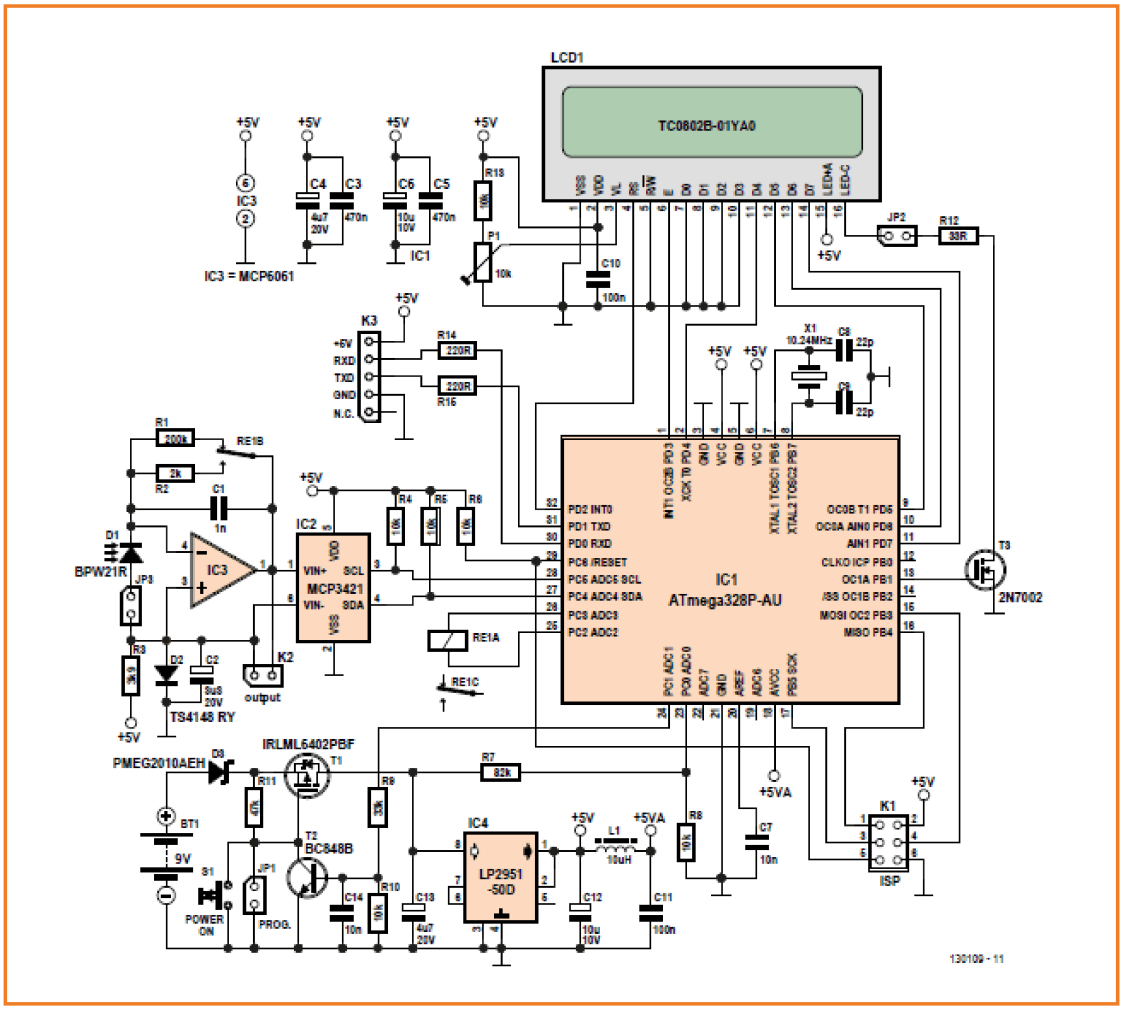


Figure 3.
The circuit of the lux meter essentially consists of a sensor, a signal amplifier, a precision A/D converter, a microcontroller and a display module.

Circuit description

At 1 lx (with RE1B in the R1 position) the voltage at the output of IC3 is approximately 1.8 mV (9 nA x 200 kΩ) That is still not enough for the A/D converter of a normal microcontroller. This could be solved by adding another gain stage, but if you want a precise instrument it's better to do the job right. Instead of using the so-so ADC integrated in the ATmega328, we opted for an external A/D converter (IC2) with an integrated precision input amplifier and a resolution of 18 bits. That may sound like overkill, and in fact we only use 15 bits, but there are good reasons for this approach.

One of the main reasons is the measuring range, since the working range of the human eye extends from 1 lx (full moon) to 100 klx (full sunlight). This corresponds to five full decades, and it requires circuitry with a very large dynamic range. This requirement is met as follows: Switching between R1 and R2 changes the signal level by a factor of 100. This means that the subsequent circuitry only has to handle a dynamic range of 1000:1. Since the internal gain of IC2 can also be switched between 1x and 8x, the resulting dynamic range is sufficient to obtain accurate measurements over the complete instrument measuring range. For the details of switching the gain levels and measuring ranges, see the firmware code. In any case, the accuracy of the circuit is approximately 0.5%.

IC2 is controlled by the microcontroller (IC1) over the I²C bus. It is used to read data and configure IC2. The relay is also controlled by two outputs of IC1, which allows the current through the coil to be reversed. This direct drive arrangement is possible because the relay (RE1) can manage with a coil current of about 24 mA. RE1 is what is called a bistable relay, which only needs

a short current pulse in one direction to switch over and remains latched in the new position without any current. It can be switched back by simply applying a short current pulse in the opposite direction. This method prolongs battery life. Relay contacts are preferable to any sort of MOSFET for this application.

IC1 also perform three other tasks. Some of its pins drive the LCD in nibble mode, and another pin switches the backlight on via T3 during a measurement session if necessary. Two other pins handle serial data transfer with TTL signal levels. The pin assignments of K3 are compatible with the Elektor BOB USB to serial converter. This gives you a serial interface over USB if you need it.

Another function of IC1 is to maintain the supply voltage (via pin 24 and transistors T2 and T1) for one minute after the device is switched on by pressing button S1. After this the instrument switches off automatically to prolong battery life.

Another remark about power: with a battery-powered instrument such as this, it's worthwhile to take measures to save energy. One of them is the previously mentioned control of the LCD backlight via T3, or disabling the backlight with JP2. Illumination is only activated automatically and adapted to the ambient light level when the ambient light level is less than 64 lx. This smart measure by itself saves about 20 mA under bright conditions, which is fairly significant considering that the current consumption of the rest of the instrument is 14 mA. To avoid the power dissipation of a standard voltage regulator, a special low-power, low-dropout regulator (IC4) is used to generate the regulated 5 V supply voltage. The regulator can work down to a battery voltage of approximately 5.5 V. At lower input voltages its output voltage collapses. The display stops working below 5 V. Choke L1 provides additional filtering for the analog portion of IC1.

Construction

When fitting components on the PCB you need a steady hand, a fine soldering tip, small tweezers, soldering paste or thin wire solder and a magnifying glass. Although the sensor is an old-fashioned leaded component, nearly all of the other components are SMDs— mostly in the especially popular 0603 package in the case of resistors and capacitors. If you are newcomer to soldering SMDs, we strongly advise that you draw on the assistance of an experienced colleague or friend. **Figure 4** shows the component side of a prototype circuit board, which should be enough to give you an idea of what you're up against. A magnifying glass is helpful when you're looking for shorts between SMD pins, which can be eliminated with a bit of desoldering braid and flux. That works fairly will with the pin spacings of the ICs used here.

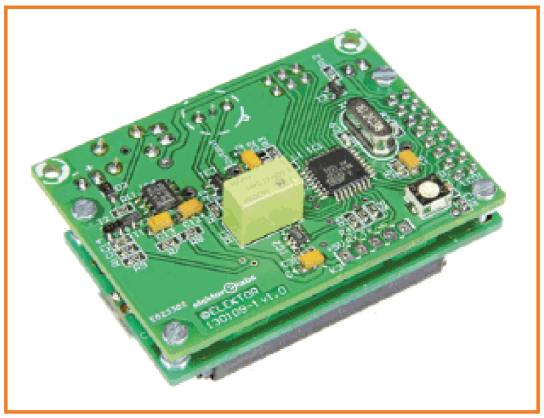


Figure 4.
The component side of the assembled prototype board. Nearly all of the components are mounted on this side, even the "tiny" SMDs in type 603 packages.

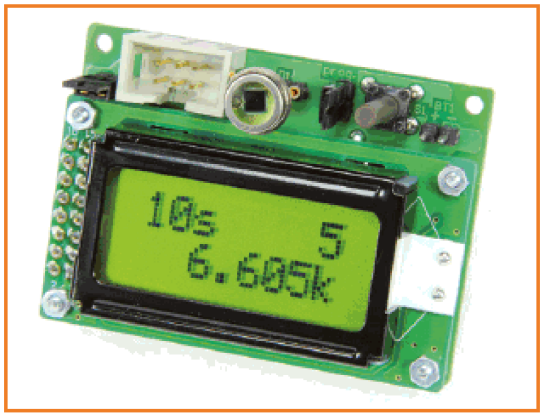


Figure 5.
Circuit board with mounted display module. The photodiode and the power-on button are at the top.

The display is mounted on the other side of the board (see **Figure 5**). If you want to mount the LCD module so it can be removed, you can implement the connection using socket headers and long pin headers. However, if you don't need to have a dismountable display you can simply use suitable lengths of wire (16 in total). It's important to keep the LCD module spaced away from board enough to avoid short circuits between the rear of the display module and the PCB.

Firmware

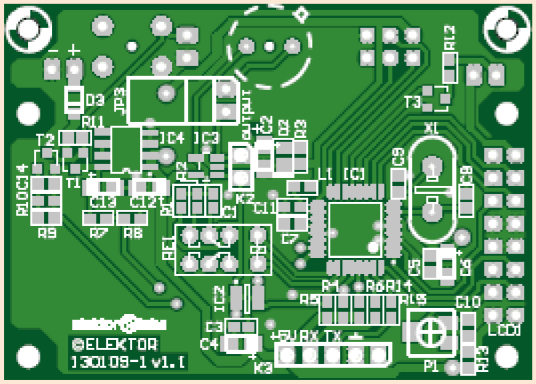
The software for IC1 was written in C using the free AVR Studio IDE (V4.18) and is available free of charge (including the hex file) from the Elektor web page for this project [4]. A 200 kHz clock

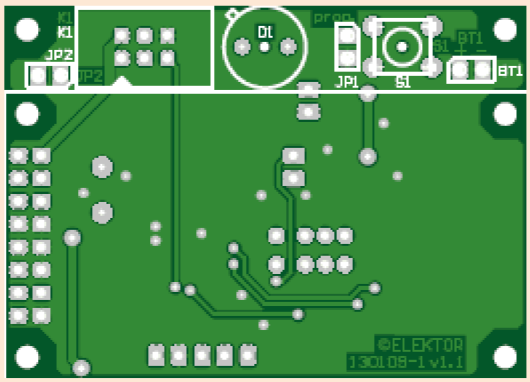
(5 ms period) for the relay drive pulse is derived from the 10.24 MHz clock signal, and from it a 2 Hz clock is derived for sequence control. The code for driving the LCD was taken from an article on the mikrocontroller.net website [5], where it is extensively commented. The I²C routine is recycled from a project for a weather station, which is why it contains unused (leftover) code for a rotary encoder and other things. If you are interested in the details, see the comments in the code.

The standard ISP connector K1 is used to download the program code to IC1. To power the microcontroller during the download process, the battery must be connected and JP1 must be fitted. Alternatively, you can keep S1 pressed for the duration. The program starts running for the first time after the download.

Since all bytes in the EEPROM memory of a new microcontroller are set to $FF and are restored to that value when the device is reprogrammed, the microcontroller can easily see whether the instrument is starting up for the first time. If the value FF_{hex} is found in two EEPROM bytes, a calibration factor of 1 is provisionally stored in those two bytes. This corresponds to a conversion ratio of 9 nA = 1 lx. More precise calibration can be performed later via the serial interface.

The next step is to measure the offset of the opamp and the A/D converter without the pho-





Component List

Resistors

Default ratings: 1%, 0.1W, SMD0603
R1 = 200kΩ
R2 = 2kΩ
R3 = 3.9kΩ
R4,R5,R6,R8,R10,R13 = 10kΩ
R7 = 82kΩ
R9 = 33kΩ
R11 = 47kΩ 5%
R12 = 33Ω 5%, 0.2W
R14,R15 = 220Ω 5%, 0.2 W
P1 = 10kΩ 20% trimpot, SMD, Vishay TS53YJ103MR10

Capacitors

Default ratings: 10%, 10V, SMD0603
C1 = 1nF 5% 50V, C0G/NP0
C2 = 3.3µF 20V, SMD case A, tantalum
C3,C5 = 470nF
C4,C13 = 4.7µF 20V, SMD case A, tantalum
C6,C12 = 10µF, SMD Case A, tantalum
C7,C14 = 10nF 50V
C8,C9 = 22pF 50V, C0G/NP0
C10,C11 = 100nF 25V

Inductors

L1 = 10µH 20%, 250mA, 1.05Ω, SMD 0603

Semiconductors

D1 = BPW21R, TO-5
D2 = TS4148 RY, SMD 0805
D3 = PMEG2010AEH, SMD SOD-123F
T1 = IRLML6402PBF, SMD SOT-23
T2 = BC848B, SMD SOT-23
T3 = 2N7002, SMD SOT-23
IC1 = ATmega328P-AU, programmed, SMD TQFP 32A
IC2 = MCP3421A0T-E/CH, SMD SOT-23-6
IC3 = MCP6061T-E/OT, SMD SOT-23-5
IC4 = LP2951-50D, SMD SO-8

Miscellaneous

K1 = 6-pin (2x3) pinheader, 0.1'' pitch
BT1,JP1,JP2,K2 = 2-pin pinheader, 0.1'' pitch (K2 optional)
JP3 = 2-pin pinheader, 0.1'' pitch, right angled
K3 = 5-pin pinheader, 0.1'' pitch, right angled
JP1,JP2,JP3 = jumper, 0.1'' pitch
S1 = pushbutton, 6x6 mm, vertical, make contact
BT1 = 9-V-battery clip with wires, optionally with receptacles for pinheader pins
RE1 = relay, bistable, 2 contacts, coil 4.5V / 202.5Ω, 1A/110VDC contacts (Panasonic AGN2104H)
X1 = 10.24MHz quartz crystal, 18pF, HC-49/4H
LCD1 = LCD module, 2x8 characters, backlit, 58x32 mm, TC0802B-01YA0 (Elektor Store # 120061-75)
PCB # 130109-1

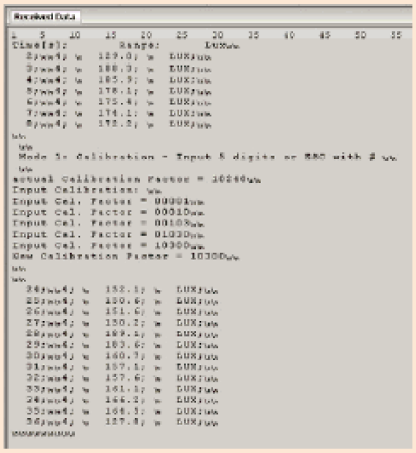
todiode (JP3 must be removed for this purpose), show it on the display, and store it in EEPROM for correction purposes.
From this point on the instrument displays a new reading every half second and provides a new value on the serial interface every second. After each measurement value output on the serial interface, a semicolon is sent as a separator, so it is easy to copy the received data to a spreadsheet on the PC. After one minute of operation, the firmware switches off the instrument.
The next time it is powered up by pressing S1, the value FF_{hex} is no longer present in the EEPROM and the instrument starts measuring brightness immediately. Of course, JP3 must be fitted for this purpose.

Miscellaneous

Pressing S1 once causes the battery voltage to be measured and displayed. After 1 second, the instrument starts measuring illuminance again. In Figure 5 you can clearly see how the measurement results are displayed. The elapsed time in seconds is shown in the top line on the left, and the measuring range is indicated on the right. The lux meter switches automatically between its six ranges. In this example, the "4" at the top right indicates that the 1 klx measuring range is selected, which matches the reading "633.2" in the bottom line.
The data transmission rate on the serial interface is 38.4 kbaud, which is a good compromise between speed and noise immunity.
To see how well the DIY lux meter performs, we compared it with a commercial instrument from Testo. Comparison with the Testo 540 model showed only minor differences of about 3 to 4% of the measured value, even without calibration. With weak illumination the meter showed 108 lx instead of 105 lx, and with bright light it showed 65 klx instead of 63 klx. In the office (with fluorescent lighting) the differences were larger, amounting to as much as 30% (1,115 lx instead of 1,600 lx). That might be due to the relatively warm tone of the light or to the fact that the light sensor of the Testo 540 is covered by a milk glass dome, so it is not clear which measurement is better.
In any case, we can say that this DIY lux meter is a versatile instrument that it can hold its own against commercial meters, particularly after additional calibration.

(130109-I)

Calibration



For calibration you can either use a reference light source at a known distance or another light meter. To take an extreme example, if the actual value is 410 lx and the reading from the lux meter is twice this value (820 lx), the calibration factor is 0.5. The calibration factor is transmitted serially as a five-character ASCII string, so the calibration factor output in this case would be "05000". Another example (more realistic): If the lux meter shows only 400 lx instead of 410 lx, the calibration factor is 1.025 (equal to 410/400). The corresponding ASCII string in this case is "10250".

After the lux meter has been switched on and connected over the serial interface to a PC where the data sent at 1-second intervals can be seen in a terminal emulator window, you can go ahead because the firmware of the lux meters waits in the background for the entry of a single character. If the entered character is "1", the firmware switches to the mode for entering the calibration factor. For this you will first see the following output in the terminal window:

```
Mode 1: Calibration - Input 5 digits or ESC with #
actual calibration factor = 10000
Input Calibration:
```

Now you can enter the previously calculated string for the new calibration factor. After the five characters have been entered, the instrument returns to normal measurement mode. The screen shot shows an excerpt from a Hyperterminal session.
The microcontroller must be reset before the new calibration factor takes effect. One way to do this is to briefly connect pins 5 and 6 of K1 together. This procedure can be repeated as often as desired. The calibration factor is retained until a new one is entered or until new firmware is downloaded to the microcontroller.

Web Links

[1] BPW21R data sheet: www.vishay.com/docs/81519/bpw21r.pdf
[2] Lux/lumen conversion: http://ledstuff.co.nz/data_calculators.php
[3] USB to serial converter: www.elektor.com/ft232r-usb-serial-bridge-bob-110553-91
[4] Project web page: www.elektor-magazine.com/130109
[5] www.mikrocontroller.net/articles/AVR-GCC-Tutorial/LCD-Ansteuerung

DesignSpark Tips & Tricks

Day #14: The Autorouter

By **Neil Gruending**
(Canada)

Today let's drop the manual PCB design work and look at DesignSpark's automatic routing tools.

If you've never used an automatic PCB track router ("autorouter") before, today is the day. We'll start by introducing the autorouter and then we'll see how to configure and use it.

What is an Autorouter?

A PCB autorouter can route some or all of your circuit board for you using a set of predefined rules. Most autorouters, including DesignSpark's, use shape based multipass technology which gives them a better chance to completely route a board. Grid based autorouters use a grid where only 1 trace or feature could occupy each location at a time but this makes it difficult to route complex boards with a lot of traces. However, a shaped based autorouter uses the actual copper shapes on the board to determine where to route the traces which allows the autorouter to pack the traces as tight as possible in a given area. The DesignSpark autorouter is also a multipass router which means it will try and route a board repeatedly until all of the routing errors are gone or the number of tries has been exhausted.
The DesignSpark autorouter is configured using the Route All Nets window in **Figure 1** which you access from the Tools → Auto Route Nets → All Nets menu. Autorouters always try to calculate the easiest or lowest cost route for a net, and adding rules controls how those routes are calculated. Just like the automatic component placement tool, the autorouter can be invoked in several different ways but is only configured in the Route All Nets menu.

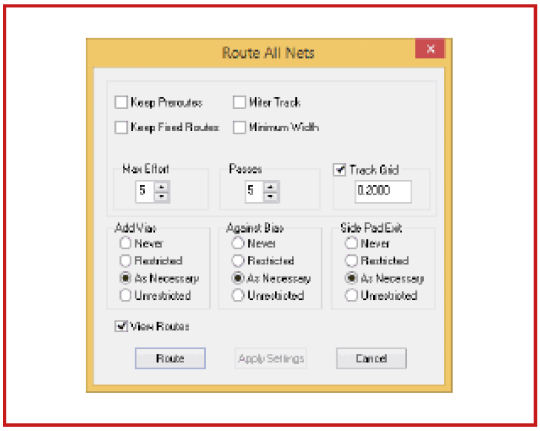


Figure 1.
The Route All Nets window.

One of the key rules is what the autorouter should do with the traces on the board when it starts. By default it will rip up or delete any traces that it needs to in order to route the board but that's usually not what you want. This is where you would use the Keep Preroutes and Keep Fixed Routes rules. A fixed route is any track segment that is marked Fixed for Router in the track segment properties and DesignSpark automatically sets that property for manually routed traces. The Keep Fixed Routes rule tells the autorouter not to alter any of the fixed route traces in any way. A preroute is a trace that the autorouter will leave intact when it starts but they could be rerouted when routing subsequent passes. Enabling the Keep Preroutes rule tells the autorouter keep prerouted trace segments when it starts.
Another important rule is the Miter Track rule. By default the autorouter will route a board with 90 degree corners but enabling the Miter Track rule instructs the autorouter try and miter as many 90 degree as possible as the last step after routing the board.

The amount of time and effort that the autorouter will use when routing the board are controlled by the Max Effort and Passes rules. The autorouter will try harder and longer as the Max Effort is increased but normally you wouldn't want to increase this effort by very much because the autorouter can usually do a better job by giving up and trying again on the next pass. The Passes rule tells the autorouter how many times it should retry routing the board. When it's set to more than 1, the autorouter will try and route the board while allowing errors like traces to overlap and then it will try and fix all of those errors on successive passes.

The Against Bias rule tells the autorouter whether or not is should follow the routing bias rules for the board. You can specify the routing bias for a copper layer using the Layers tab in the Design Technology window to either X, Y or None. An X bias tells the autorouter to try and route traces horizontally as much as possible and a Y bias is used to route traces vertically instead. Normally you don't want to specify a bias because it gives the autorouter the most flexibility when routing the board.
I recommend enabling the Track Grid rule and setting it to your desired routing grid. The autorouter doesn't need to use a grid while running but it will make the board much easier to cleanup by hand later. I also recommend leaving the Add Vias and Side Pad Exit rules set to As Necessary.

Using the Autorouter

Now let's take a look at the different ways how DesignSpark autorouter can be used to route boards. The command to route an entire board is Tools → Auto Route Nets → All Nets. **Figure 2** shows what the LED driver board from last week looks like after being routed. It needs some cleanup but that's typical for an automatically routed board. But how did the autorouter know what track and via styles to use while routing?
By default the autorouter will chose its own settings for vias and tracks but you can override that by using net classes like in **Figure 3**. A net class is how DesignSpark organizes nets into groups so that you can use the same routing settings for all of the nets in that group all at once. By default DesignSpark will create one named signal and put all of your nets into it, but you can have as many classes as you want. Net classes are created and edited in the Net Classes tab and you associate nets with net classes in the Nets tab. In my case I set the signal net class to use the track style 'Signal' and the via type 'SignalVia'. DesignSpark has other autorouting modes as well, like routing an individual nets or just a net class. But I think that one of the most useful features of the autorouter is its ability to fan-out all of the power connections on a component using short stub traces and vias if you have power planes in your board. All you have to do is tell the autorouter to only route the plane nets and it will save you a bunch of time. It's also possible to use the autorouter interactively by selecting nets or components on the board, right clicking on them and then choosing Auto Route. The autorouter will then route only the selected items.

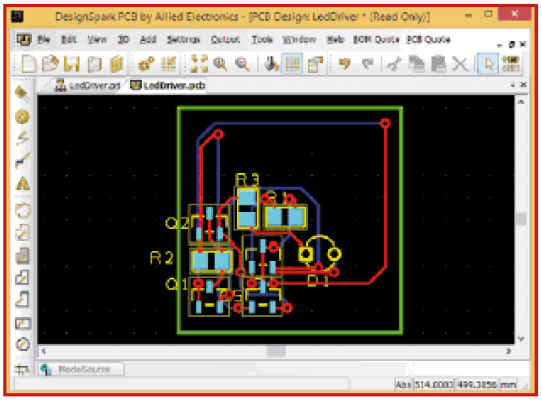


Figure 2.
Routing the LED Driver board.

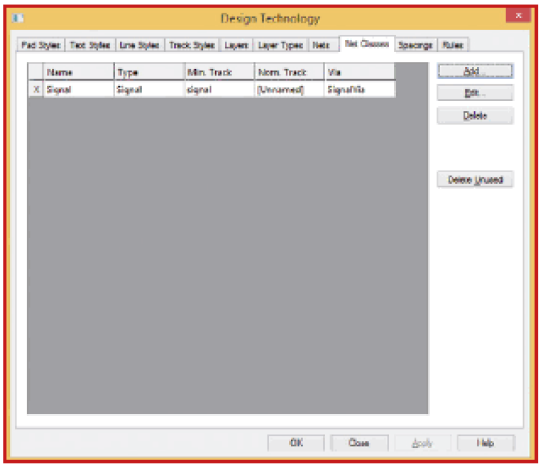
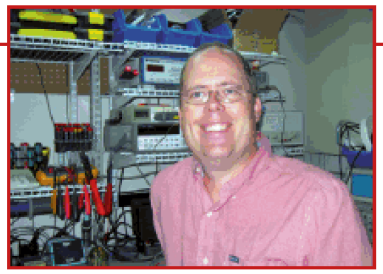


Figure 3.
The Net Class window.

Conclusion

Today we explored DesignSpark's autorouting tool and it can be a real time saver when routing a board. I usually like to route boards completely by hand but the DesignSpark autorouter is flexible enough to take away much of the tedious routing work. Next time we'll look as some of DesignSpark's other PCB features.

(140242)



Neil Gruending has used numerous different PCB CAD packages as an electronics design engineer over the years. Neil is pretty particular about his tools and likes to learn how to maximize his productivity with them whenever possible. He also enjoys sharing what he has learned on his website at www.gruending.net and on Twitter as @ngruending.

Magnetrons
Weird Component #8

By **Neil Gruending**
(Canada)

I've been talking about semiconductor devices for the last few installments so I thought I would change things up a bit and talk about magnetrons. You're familiar with them already because they're the key component in microwave ovens, but they are also used as heating, drying and curing elements for semiconductor manufacturing and other industries. So what are magnetrons and how do they work?

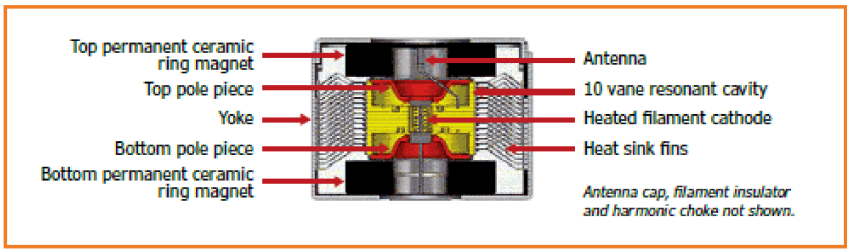


Figure 1.
Magnetron structure. [1]

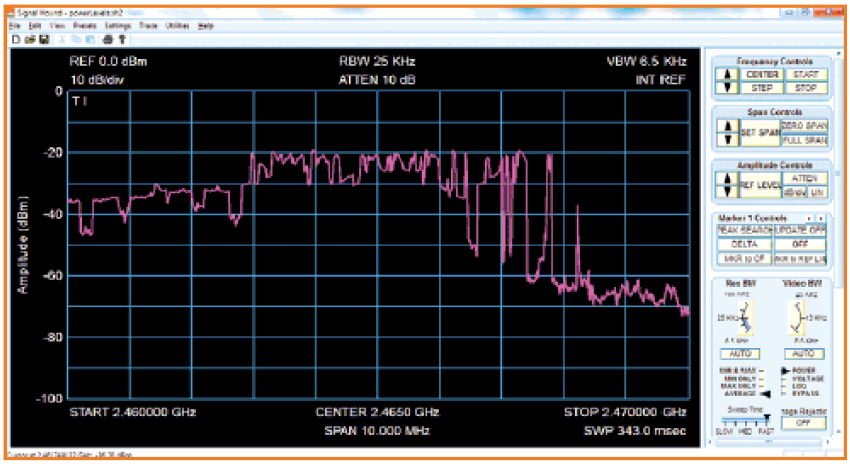
The drawing in **Figure 1** shows how cavity magnetrons are constructed. The cathode in the middle of the magnetron is heated causing electrons to flow into the cavity. Normally they would flow directly to the anode and form a diode but the upper and lower magnets force the electrons to flow around the cavity in circles and accelerate until they can reach the antenna anode and exit the magnetron. There's also two resonant cavities inside of the magnetron at each end which tunes the RF frequency that comes out of the antenna. One of the first applications for magnetrons was as a radio energy source for radar systems. Magnetrons were one of the most efficient ways to generate the short high power radio pulses required for radar imaging. One challenge of using magnetrons is that their output frequency will change with temperature and output impendence. Another issue is that the magnetron outputs a wide range of frequencies simultaneously which means that a radar receiver can't be very selective about the frequencies it receives which can reduce its sensitivity. But the magnetron's output power and efficiency far outweighed these downsides.

Commercial magnetrons need to be driven with an anode voltage in the 4 kV to 10 kV range depending on the model. Smaller models in the 1 kW to 2 kW range are typically used in microwave ovens and are air cooled. Larger industrial 3+ kW models are water cooled to keep them from overheating.

Old discarded microwave ovens are a great way to get some magnetrons to experiment with. Just remember that microwave radiation can hurt you and that they have supply voltages in the 1000's of volts which can also be lethal. I dug one out to see if I could measure the frequency drift of a magnetron over time, and the result is shown in **Figure 2**. It was quite a bit harder than I expected because the magnetron was changing frequency faster than the spectrum analyzer could measure in real time so I used a peak detection mode to try and see how much the frequency would vary. For this test the frequency started at about 2.465 GHz and then drifted for the duration of the 30-second test. The line at –20dB shows the frequency deviation and would probably be much more flat if I had run the test longer. Note that another challenge when working with magnetrons is how to dissipate all of the RF power. For this test I used a sealed waveguide setup with a water cooled load element. If you do decide to play with a magnetron, just remember to play safe.

(140241)

Figure 2.
Magnetron Output Spectrum.



Web Link

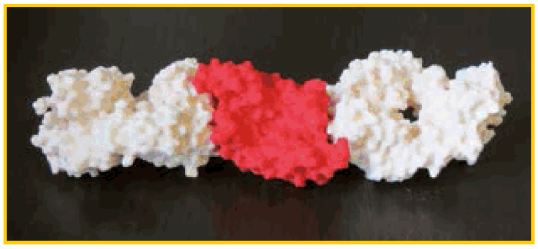
[1] www.cst.com/Applications/Article/Magnetron-And-Microwave-Oven-Design-To-Solve-Wi-Fi-Interference-Issues

3D Printing Sure Can Be Useful

Or how to fight cancer with cancerous materials...

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

In the June 2014 edition of this column I expressed my doubts on the usefulness of 3D printing, or, to be more precise, on the usefulness of the objects produced by 3D printer enthusiasts, and I asked you to send me examples of home-made 3D printed things that you believe are useful. Some people took up the challenge and sent me photos of (and links to) the fruit of their work. This showed me two things: first of all, the word "useful" does not mean the same to everybody and second, the diversity of applications for 3D printing is huge. Have a look at the entries posted on Elektor.Labs [1] and see for yourself how our members use 3D printing to restore World War I aircraft, create realistic models of everyday objects for model trains or to make tools and custom parts for their projects.



3D printed proteins. The red one is the enemy molecule produced by the cancer cell, the white proteins try to catch the red one.

The silicon keypad created by François-Xavier Dufour.

Useful, but on a limited scale at best.
Recently at the MakerSpace56 Fablab in Vannes (France, not far from where I live), where 3D printing seems to be one of the main occupations, I came across an application that was new to me: 3D printing of molecules. Not at their real size of course, but enlarged models of molecules. The goal was to visualize a protein that is (going to be) used to trap another protein produced by cancer cells. Now this is what I call useful, using 3D printing to create something that can help solving a problem that touches many people. A few months ago I assisted at a 3D printing contest/event organized by RS Components where the goal was to invent just such a 3D printable object. The outcome of this event was, I felt, a bit disappointing and I am sure that had this anti-cancer protein had been presented here, it would have beaten the competition hands down. As did the RS event and many other 3D printing attempts that I have witnessed, the protein project also showed that 3D printing still has a long way to go before it will be the common household tool that the experts say it will once be. For now 3D printing is expensive, very slow and more often goes awry than right.
In the article mentioned at the beginning I also promised a prize for what I felt would be the best submission. Since the protein did not participate in this mini contest, I elected François-Xavier Dufour as the winner. He showed (me) how he uses 3D printing to create molds for silicon keypads [2]. The usefulness of this application is of course debatable, but I found the idea refreshing of using 3D printing as an intermediate step in the process of making an object instead of as the final step. Also, I like rubber keypads. Congratulations, François-Xavier!

(140048)

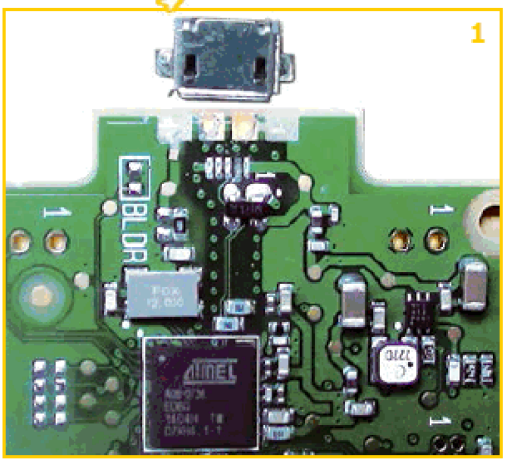
[1] www.elektor-labs.com/node/4056
[2] www.elektor-labs.com/node/4104

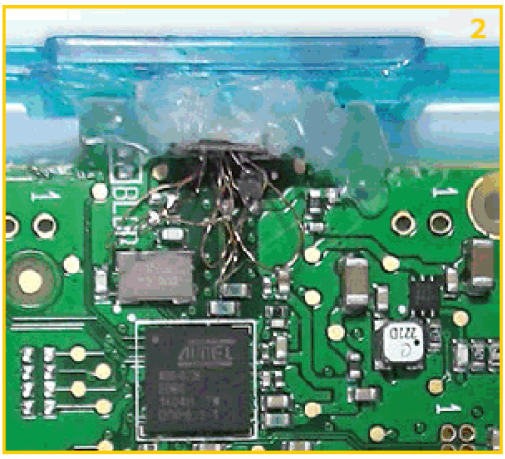
USB Fix

By
Wolfram Pioch
(Germany)

Elektor reader Wolfram recently received on loan an Atmel ICE Programmer/Debugger to review (see elsewhere in this edition). Testing did not go as smoothly as he hoped. He wrote to us:

The second time I pushed the micro-USB plug into the Atmel ICE box it went in very easily, way too easily! In fact there didn't seem to be any resistance to the plug at all. A closer look showed that the socket in the case opening was missing. I am usually very careful handling equipment on loan but it looks at though I have managed to break the connector just with gentle pressure.
Things are not looking good, I am supposed to be reviewing this piece of kit and I've busted it already before I've even had a chance to turn the coffee machine on. The obvious solution would be to get on the phone and arrange a replacement but my hacker instincts kick in and I reach for a screwdriver. Inside wasn't a pretty sight (**Figure 1**); the connector had become detached and taken with it the earth pads on either side of the connector and the track to the second pin from the right in the photo. The big earth pad under the connector had no solder on it at all, just unmelted spots of adhesive put there during the component mounting stage. Looks like a reflow failure.





In any case the fixings just couldn't withstand the pressure of inserting the plug. If the remaining good PCB tracks had been properly attached to the connector pins they too would probably be ripped off by the plug when I pushed it home.
Worth the hassle of repairing? As it stands the board is pretty much useless but with a hot soldering iron and a few lengths of wire I think I can probably resurrect it.
The repair took a bit longer than I expected. The connector pins are so close together that even with the thinnest soldering bit it's difficult to make a joint without soldering the neighboring pins as well. With all the wires connected to the socket I gave each a gently tug to make sure the joint was secure. Two came adrift so I had to repeat the whole process again: first desolder, clean and then resolder the five wires in a sequence from left to right. With all five secure I took a good look with a magnifying glass to make sure there were no obvious shorts.
Now to line up the socket in its original position it was necessary to put a 180° bend in the wire which of course resulting in another wire breakage and resolder sequence. Now I refit the PCB and connector back into the case which resulted in… you guessed it. All I can say is by the end I got pretty good at soldering tiny bits of wire onto tiny connector pins. The connector was eventually fitted into its original position using lots of hot glue to hold it firmly to both the case and PCB (more on the case than the PCB). I admit the finished repair (**Figure 2**) doesn't look too pretty. Call me superstitious but I know if I make an effort to tidy it up and get it looking neat the greater the chances are I will need to take it apart again for repair.
To test out the mend I didn't want to risk damaging the PC's USB port so I gingerly plugged the cable from the ICE into a USB hub connected to the PC. When I powered it up there was a bright red glow, not from the board or the hub but from the power-on LED. I hooked up the target system with the ICE cable and it all worked fine.
Once it was back in its case I fixed the cable to the case with some more hot glue to reduce strain on the connector. I thought it best to stick a label to the side of the case with the warning "Use at your own risk!"
How long is the fix likely to last? For sure it has already lasted longer than it did in its original state. Over coffee I began to wonder why the Atmel ICE is also available individually ('PCBA Kit') and why Atmel Studio 6.2's Help file gives details on opening the case…

(140274)

Chip Tip:

MagI³C-VDRM

A voltage regulator and then some...

By
Viacheslav Gromov
(Germany)

What are your first thoughts when someone mentions voltage regulators? Maybe you think of voltage stability problems or spurious electrical noise or the extra circuitry you need to add to protect the regulator from external influences. Maybe you think how their design has matured over the years: regulators are now smaller, better designed and more efficient. The process is ongoing and there is still room for improvement especially with respect to their low-power performance. Here we take a look at an example of a new family of step-down converters based on the VDRM concept (**V**ariable Step **D**own **R**egulator **M**odule).

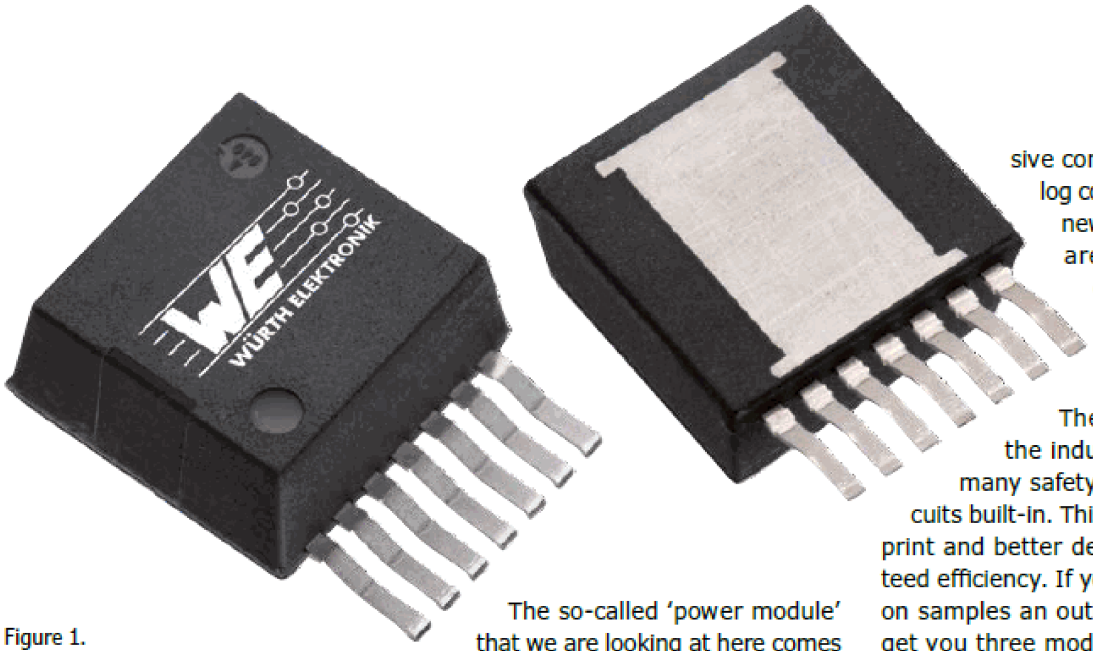


Figure 1.
The regulator uses a TO263-7EP package.

The so-called 'power module' that we are looking at here comes from the German company Würth Elektronik. The company has already earned a good reputation for producing high quality passive components and books on analog component technology. Würth's new family of voltage regulators are not what you would call cheap, but they add weight to the old adage that you only get what you pay for.

The regulator ICs incorporate the inductive components and have many safety features and protection circuits built-in. This leads to a smaller PCB footprint and better design reliability with guaranteed efficiency. If you are keen to get your hand on samples an outlay of around $35 / €25 will get you three modules of your choice. For test purposes and evaluation the company has also made available an evaluation board which retails in the higher price bracket.

Table 1. Data on the different types.

Type	V_{in}	V_{out}	I_{out}	P_{out}	Switching frequency	Internal inductance	Evaluation Board Part No
WPMDH1200601JT	6 to 42 V	0.8 to 6 V	2 A	12 W	0.2 to 0.8 MHz	10 µH	178 020 601
WPMDH1102401JT	6 to 42 V	5 to 24 V	1 A	24 W	0.2 to 0.8 MHz	15 µH	178 012 401
WPMDM1500602JT	6 to 36 V	0.8 to 6 V	5 A	30 W	0.812 MHz	3.3 µH	178 050 601
WPMDH1152401JT	6 to 42 V	5 to 24 V	1.5 A	36 W	0.2 to 0.8 MHz	15 µH	178 012 402
WPMDH1302401JT	6 to 42 V	5 to 24 V	3 A	72 W	0.2 to 0.8 MHz	10 µH	178 032 401

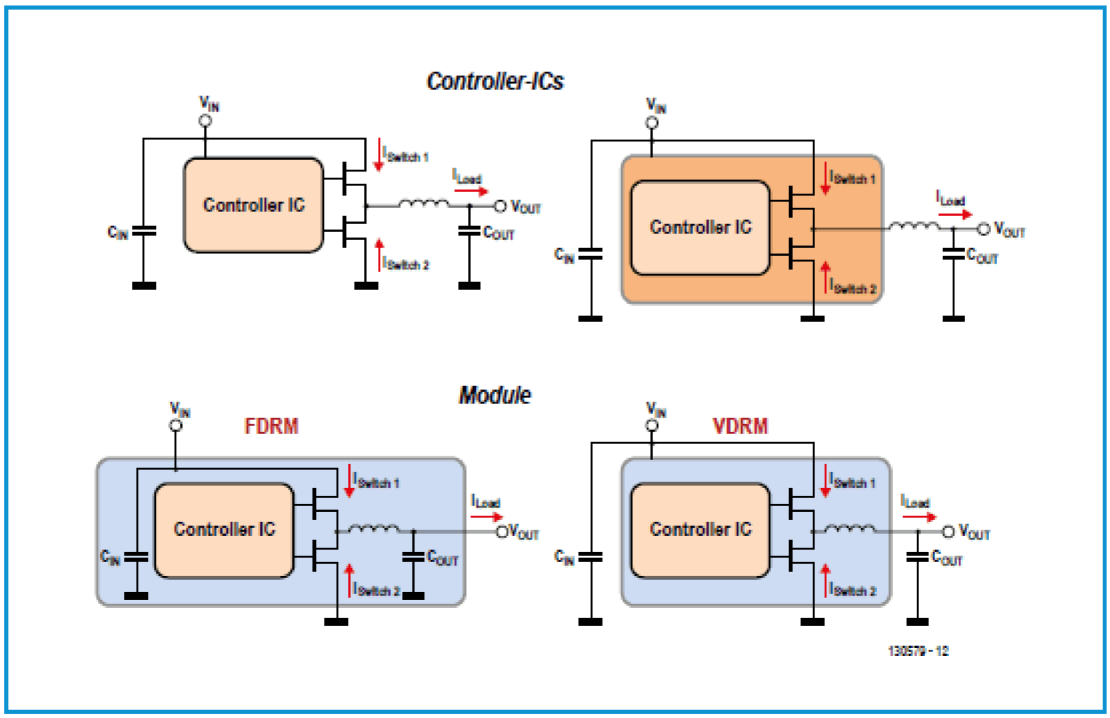


Figure 2. The difference of FDRM and VDRM to normal regulator ICs and modules.

Meet the family

The family of regulators provides solutions to cover a wide range of power and voltage requirements (see **Table 1**). There should be a version suitable for almost all circumstances. A free brochure is available from Würth [1] containing more information to help choose the optimal version for your application, altogether there are five different types available, all using the same TO263-7EP package outline (**Figure 1**). The regulator's efficiency peaks at 97% but is, amongst other things dependant mainly on load. The main difference between this family and the FDRM devices (**F**ixed Step **D**own **R**egulator **M**odule) apart from the obvious fact that these have an adjustable output voltage, is mainly that they have integrated input and output capacitors within the package (see the overview in **Figure 2**). Each device incorporates circuitry to protect against over-voltage and under-voltage as well as excessive output current and operating temperature above 165 °C. All of these mechanisms contribute to make for an uncomplicated integration into your design.

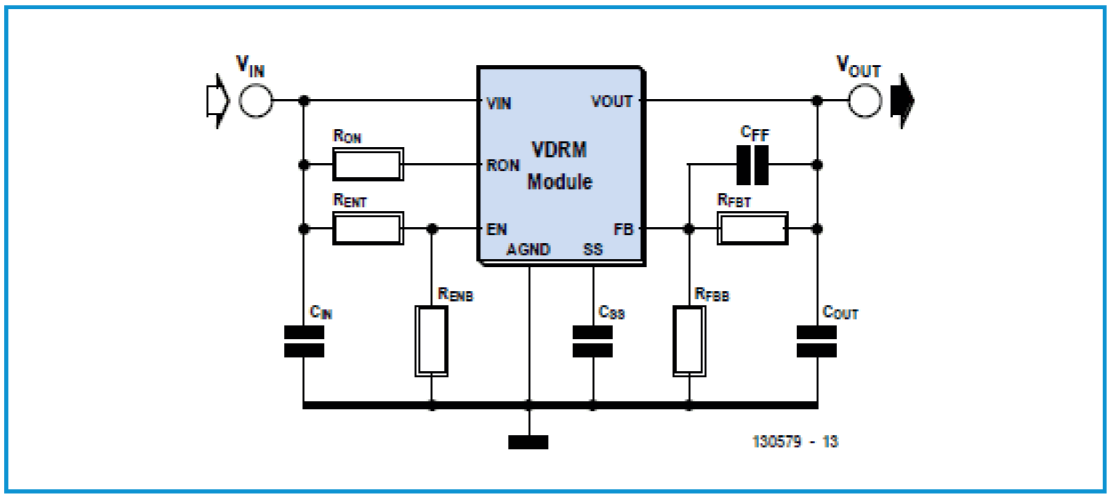


Figure 3. Internal circuit of a VDRM (excluding the WPMDM1500602JT).

The insides

The VDRM type of regulator can be configured with a voltage divider network (see **Figure 3**) R_{ENT} and R_{ENB} at the EN input determine the lower threshold below which the circuit shuts down. This UVLO feature (**U**nder **V**oltage **L**ockout) offers protection, for example to rechargeable batteries that could be damaged by discharging them too deeply. The value of capacitor C_{SS} at input SS controls the soft-start behavior. The regulator's switching frequency can (with one exception) be set by the value of resistor R_{ON} at pin RON in the range from 0.2 to 0.8 MHz.

The exception is the type WPMDM1500602JT, which allows many regulators to be clocked from the same source. On this model the RON pin has been replaced by the SYN pin to which an external clock in the range of 0.65 to 0.95 MHz can be connected. With this pin tied to ground the chip defaults to an internal clock of 0.812 MHz. The output voltage is defined by resistor values R_{FBT} and R_{FBB} which form a voltage divider network at feedback pin FB. The data sheet [1] includes a formula to calculate their values and a table of resistor values together the generated output voltage.

And the outsides

The regulator can operate at a temperature ranging from -40 to 125 °C. Switch regulators require suitably sized capacitors at their input and output to help filter and reduce electrical noise. Just like other types of regulator the capacitors used should have a small ESR value. Multilayer ceramic with X7R- and X5R dielectric are recommended as are tantalum capacitors. As you can see in Table 1, the input voltage range and switching frequency (ignoring the exception referred to above) are all the same. Referring to the block diagram in Figure 2 you can see how the internal inductors and the external capacitors are connected. The internal MOSFET transistors (which on conventional switch regulators would be external devices) are also shown.

The data sheet includes tips on PCB layout which you should pay attention to if you want the power supply to generate the lowest levels of electromagnetic interference. It is important that the earth pad on the regulator makes good contact with ground and that the feedback conductor is as short as possible. When your application calls for standard supply voltage levels you can use the FDRM variants which have a fixed output voltage and smaller package outline and which also incorporate the input and output capacitors.

The company also produces evaluation boards so that you can quickly start to explore the properties of these devices in more detail on your test bench (**Figure 4**).

(130579)

Web Link

[1] http://katalog.we-online.de/de/pm/MagIC-VDRM

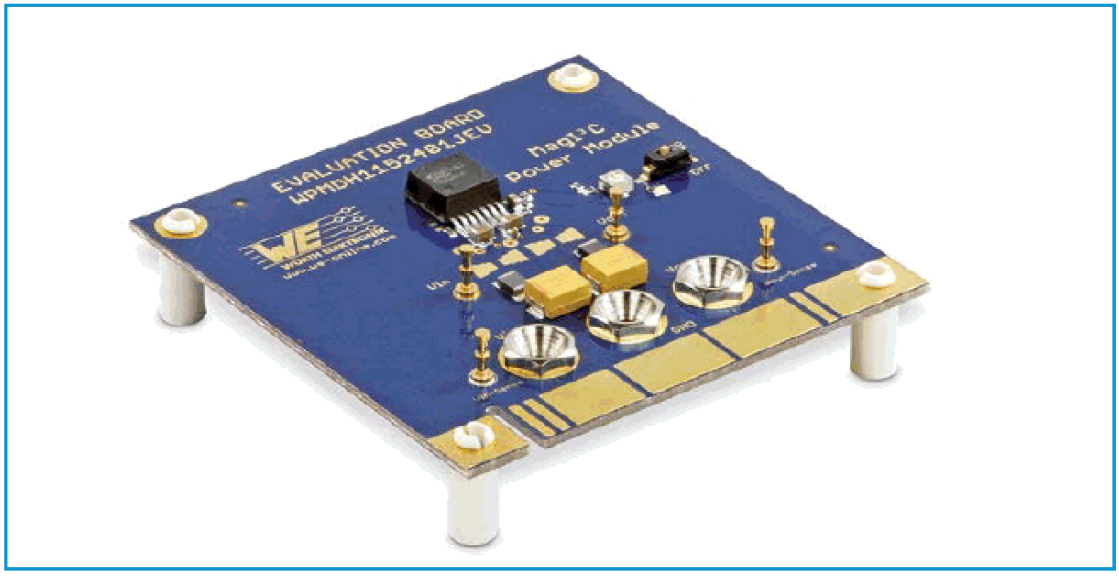
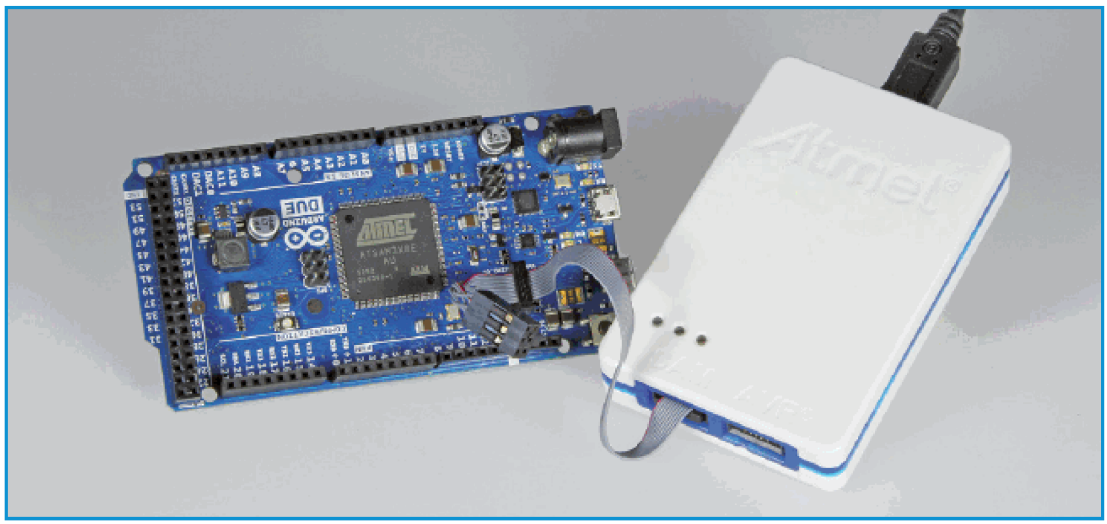


Figure 4.
The evaluation board.

One for All

The new Atmel-ICE Debugger/Programmer



By
Wolfram Pioch
(Germany)

At the last Embedded World Expo Atmel announced a new low-cost Programmer/Debugger suitable for the full range of AVR controllers including Xmega and the ARM-Cortex based controllers. We took a closer look at this interesting little box.

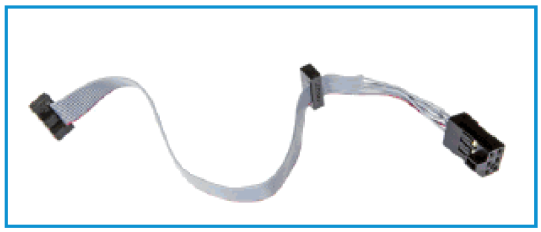


Figure 1.
The basic variant is supplied with a cable fitted with a 10-pin (.05 inch pitch) and 6-pin box-header.

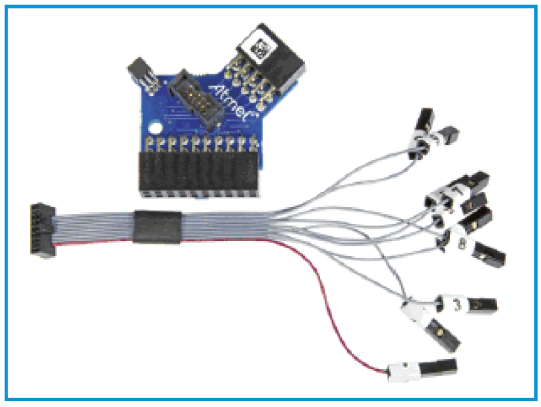


Figure 2.
The 'Full Kit' includes this Squid cable with individual sockets and an adapter board.

The Atmel-ICE (In Circuit Emulator) is the latest Programmer/Debugger from Atmel. It can't disguise its heritage; the cased version has the same dimensions (85 x 55 x 16 mm), as the existing JTAGICE3 AVR debugger. Connection to system under test is via a ribbon cable which connects to the Atmel ICE via a 50-mil pitch, 10-pin box header. The other end has connection options which allow it connect to AVR development and prototyping boards.

The Atmel-ICE is suitable for use with more processors than the standard Atmel ATmega/Xmega/ATtiny range. The company has already demonstrated how its development software 'AVR Studio' has now integrated support for proprietary ARM-Controllers (hence the name change from 'AVR Studio' to 'Atmel Studio').
The Atmel ICE is now a 'one size fits all' solution for microcontroller software development. A second 10-pin ('SAM') connector provides

programming and debugging support for ARM controller systems such as the SAM3X8E (ARM Cortex M3) used on the Arduino Due board. A comparison with the alternative Atmel debugging tools is given in the table. When programming ARM-based controllers it can be seen that the maximum clock frequency of the Atmel-ICE is significantly less than the alternative SAM-ICE debugger.

Adapter

The Atmel ICE comes in three variants: The PCBA kit is the board without any case, the Basic Kit has a case and a ribbon cable and the Full Kit has the ribbon cable, an adapter board and a flying lead mini 'squid'. We were fortunate enough to be supplied with one of the first examples of the kit by the German distributor Ineltek [1]. The full kit retails at just over £60 and the most basic board version costs about a third of that, all variants are stocked by Newark/Farnell.

The basic kit is supplied with a ribbon cable (**Figure 1**). This has two 50 mil (1.27 mm) pitch 10-way box header connectors. This fits, for example the debug pin header on an Arduino Due. At the cable-end six of the ribbon leads are connected to a six-way (100 mil pitch) female box header which connects to the popular SPI connector used by many Atmel processor boards. It is also usable as a PDI connector on Xmega boards.

The full kit includes an adapter board and another ribbon cable (Figure 2) with 10 individual socket headers (the Squid Cable). The board has a 20-way socket header (0.1 inch pitch) for a SAM-JTAG or SWD interface (like the SAM-ICE). Apart from this there is the familiar 10-way JTAG connector (0.1 inch pitch) for AVR and also a 6-way connector with ISP, PDI, debugWire and aWire signals. The pins on this connector are pitched at 0.05 inch (1.27 mm) which makes for a small programming/debug interface on small PCBs.
An overview of all the possibilities with the new tools can be found at [2].

The Atmel ICE connects to a PC via a USB cable. During testing the micro-B USB connector on the board broke away and we had to make a quick DIY repair (see separate article).
This seems to be a general problem with micro USB sockets which are usually just soldered into position. A normal sized USB-B socket would certainly stand up better to the rigors of a typical test and development environment.

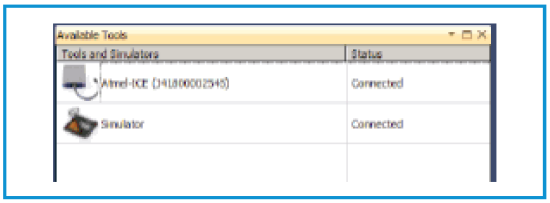


Figure 3.
Tool selection.

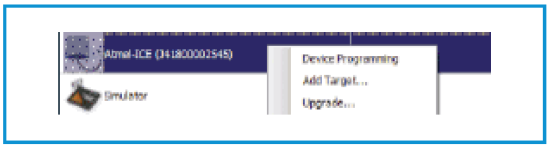


Figure 4.
Check the firmware version with 'Upgrade'.

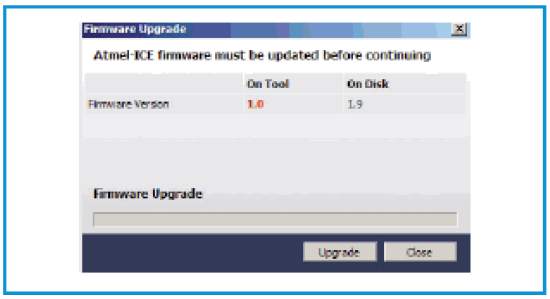


Figure 5.
The firmware needs updating.

Software

The new Atmel-ICE is supported by version 6.2 of Atmel studio. This latest version [3] must be installed in the PC before the Atmel-ICE is first used. Now when the emulation USB cable is plugged into a free PC port it will be automatically recognized and the correct USB driver will install. In Studio 6.2 you can see that the ICE has been recognized by the software.

Under '*View*' '*Available Atmel Tools*' open a window as shown in **Figure 3**. This allows you to check that the ICE is running the latest firmware version. A right click on the mouse produces a pop up menu where you can select 'Upgrade' (**Figure 4**). When the ICE firmware is not the latest version a message **Figure 5** indicates that it needs to be updated before continuing.
Before you can think about debugging the software in a project you need to first check that the ICE is communicating with the processor. We tested the ICE on three systems using different processors.
First off we hooked up an ATMega32 on an STK500 development board (**Figure 6**). Using the basic cable the Atmel-ICE is connected to the

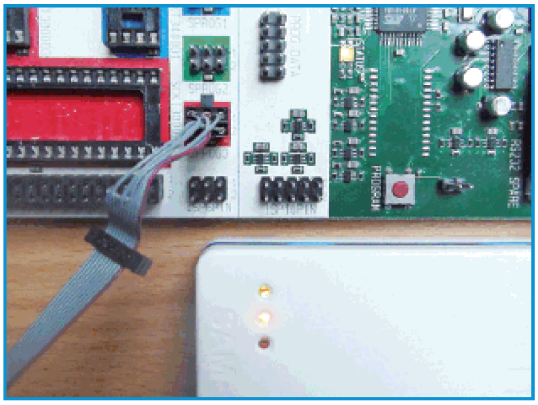


Figure 6.
ATmega32 programming with an STK500 board.

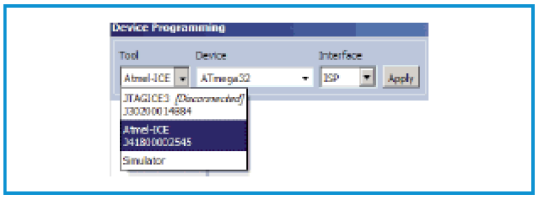


Figure 7.
First select the Atmel-ICE...

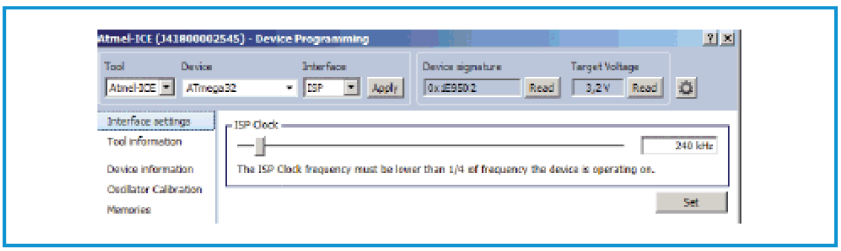


Figure 8.
... then the processor and interface.

processor–dependent pin header of the STK500. The ribbon cable is arranged so that the red wire (pin 1) fits to the side marked '1' on the STK pin header with the keyway of the 6-way connector uppermost.
In Atmel Studio in the programmer we selected 'Available Atmel Tools' 'Atmel ICE' 'Device-Programming'. Alternatively you can click on the button with a picture of a chip and lightning bolt in the tool bar.

Now with the '*Device Programming*' window open click on '*Tool*' to select the Atmel-ICE option (**Figure 7**) and then choose the target processor under *Device* (we chose ATmega32) and ISP for the *Interface*. Once these options have been selected, click the '*Apply*' button. Next under '*Interface Settings*' in device programming (**Figure 8**) you can select the ISP clock to be around 240 kHz; this should not be any faster than one quarter of the processor clock frequency. Don't forget that a brand new processor fresh from the factory will be initially configured to run from its internal RC oscillator at around 1 MHz.

An ISP clock of 240 kHz will therefore be valid and later, when the fuses have been configured to choose a faster processor clock rate, the ISP clock can also be increased correspondingly.
To make first contact with the processor click on '*Read*' under '*Device signature*' (**Figure 9**). When this produces no error message you can be confident you have selected the correct processor.

Xmega

Next in line for testing is the Xmega192A3 fitted to an STK600 board. This uses the PDI interface. The box header on the ribbon cable is fitted to the SPI/PDI pin header (**Figure 10**). The programmer set up is handled the same way as in the previous ISP set up only this time the 'PDI' interface option is selected. The PDI clock frequency has a maximum frequency of 7.5 MHz. The Xmega can also be programmed via a JTAG.

Main features of the Atmel-Debugger range

	AVR ONE!	JTAGICE3	Atmel-ICE	SAM-ICE
Interface	SPI,PDI,TPI debugWire JTAG	SPI,PDI,TPI debugWire aWire JTAG	SPI,PDI,TPI debugWire aWire JTAG/SWD	 JTAG/SWD
SPI Clock	32 kHz – 32 MHz	8 kHz – 1.8 MHz	8 kHz – 5 MHz	
JTAG Clock	32 kHz – 32 MHz	32 kHz – 15 MHz	32 kHz – 7.5 MHz	
PDI Clock	32 kHz – 32 MHz	32 kHz – 10 MHz	32 kHz – 7.5 MHz	
dW Baudrate	not applicable	4 kbit/s– 0,5 Mbit/s	4 kbit/s – 0.5 Mbit/s	
aWire Baudrate	not applicable	7.5 kbit/s – 7.5 Mbit/s	7.5 kbit/s – 7 Mbit/s	
SWD Clock			32 kHz – 2 MHz	0 – 8 MHz

For this option the Atmel ICE cable is plugged in to the JTAG connector on the STK600 card (**Figure 11**). This time we choose the 'JTAG' option and this gives us a maximum clock speed of 7.5 MHz.

Finally we tried the Atmel ICE on an Arduino Due board (ATSAM3X8E). The basic ribbon cable is used here, plugged into the 10-pin debug interface with the red identifier cable nearest the center of the Arduino board (**Figure 12**). The other end plugs into the SAM connector on the ICE. Accidentally plugging it in to the AVR connector will not cause any damage. The green LED on the debugger lights up when the cable is correctly fitted. The ICE connector will not hinder the fitting of shields to the Arduino board.

The processor type ATSAM3X8E for the Arduino Due is specified in the Interface-Settings of the Programmer. Note that the 'SWD' (Serial Wire Debug) option must be used here, the 'JTAG' will not work. The maximum clock frequency is (unfortunately) just 2 MHz.

Debugging

There is now nothing stopping us debugging a finished AVR or ARM project. Now we get to test the new SAM interface. As a first try out we recommend using a suitable example from the Atmel Studio and adapting it to your needs. Here we will demonstrate the procedure using an Arduino Due. We won't use an Arduino Sketch but instead a 'pure' C program. In the next installment we go on to show how Atmel Studio is used to develop and debug an Arduino Sketch.

To start off we select '*File*' '*New*' '*Example project*'. A window opens where we select '*SAM3, 32-bit*' as a *Device Family* and '*Applications*' in the *Category* textbox. In the list of options now available we click on '*Getting-Started Application on SAM - Arduino Due/X*' (**Figure 13**). Further options below allow you to change the project name and the location. We keep the default name and select the directory as '*D:\ATMEL ICE*'. Now confirm the selections with 'OK' and check you agree to the license terms and conditions, now Atmel Studio announces that project has been created.
To the right we see the Solution-Explorer (**Figure 14**), it shows the project structure. A click on the 'main.c' entry opens the corresponding file.

A right mouse click on the project name '*GETTING-STARTED1*' or in the main menu '*Project*' '*GETTING-STARTED1 Properties*'. (ALT-F7) opens the project properties page.
Now with '*Device*' tab the type of processor can be changed and the '*Tool*' tab allows you spec-

Figure 9.
It's a good sign when you are able to read the Device signature.

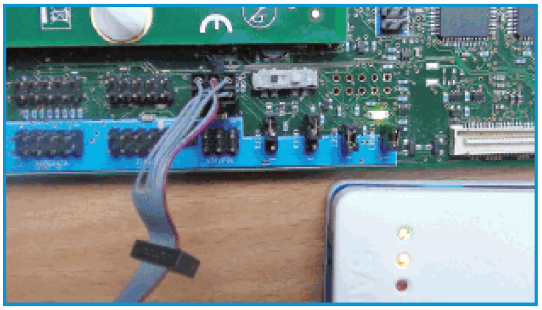
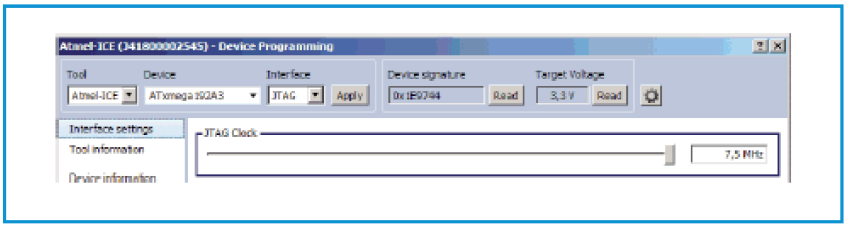


Figure 10.
Xmega programming via PDI.



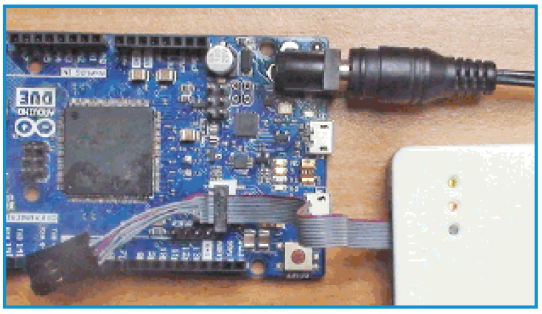


Figure 11.
The Xmega can also be programmed via JTAG.

Figure 12.
Arduino Due with the Atmel-ICE.

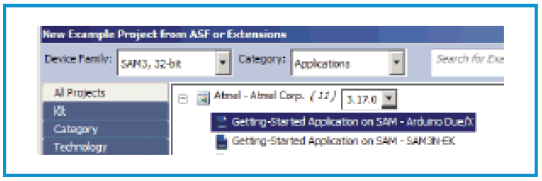


Figure 13.
Choose an application example.

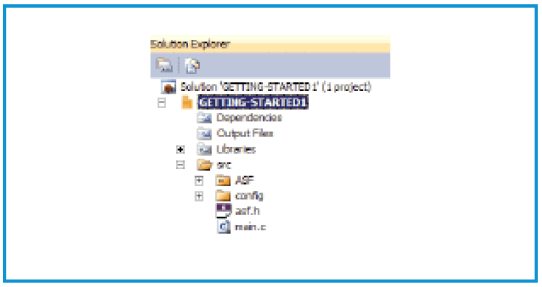


Figure 14.
Files for use in Solution Explorer.

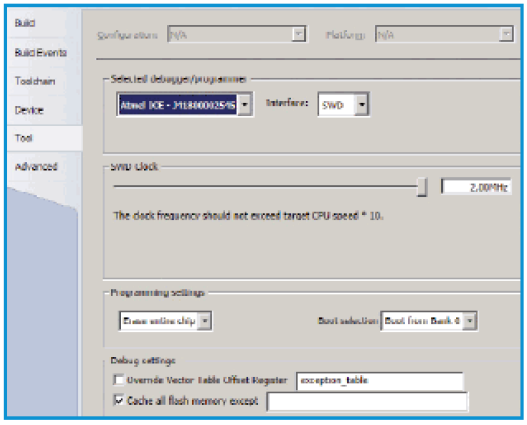


Figure 15.
Select the 'Tool' and debugger interface in the properties menu of the application.

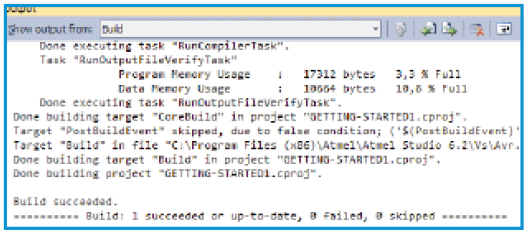


Figure 16.
The output window.

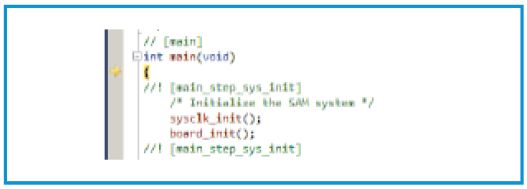
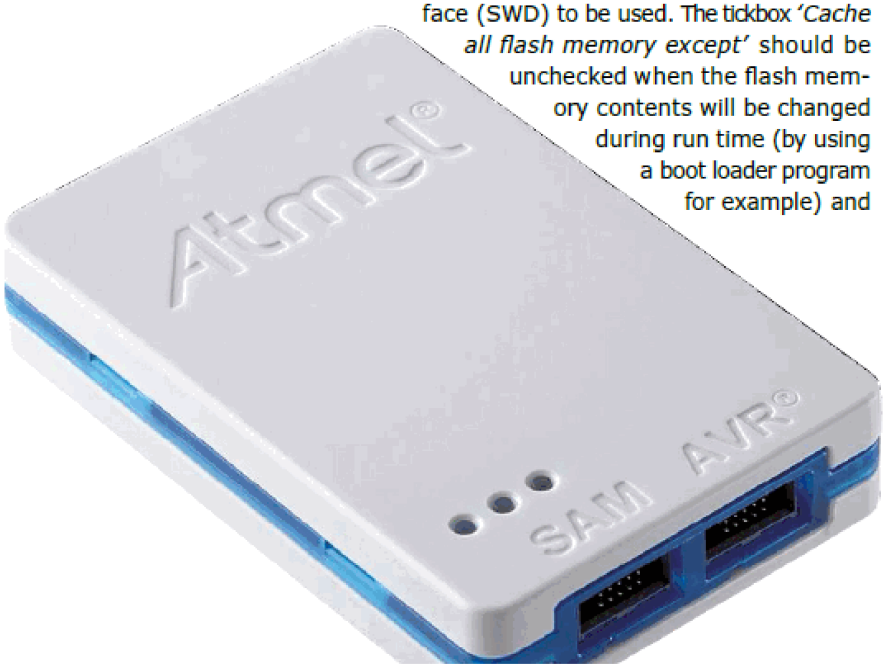


Figure 17.
ALT-F5 halts the program at the first line.

ify the type of debugger (Atmel ICE) and interface (SWD) to be used. The tickbox *'Cache all flash memory except'* should be unchecked when the flash memory contents will be changed during run time (by using a boot loader program for example) and you want to observe this. Otherwise you will always see the state of the flash memory in the memory window at the time of the program start.
All the other settings including those in other windows should not be altered at this point.
The set up can now be saved using *'File'* → *'Save Selected Items (Ctrl+S)'* or *'File'* → *''Save All' (Ctrl+Shift+S)* from the main menu.
Using '*Build*' '*Solution*' (or F7) will compile the program. We haven't made any changes to the code so we aren't expecting any errors and this is confirmed in the output window **(Figure 15)**.
Now with ALT+F5 the ELF file, that you can see in Solution-Explorer under '*output*', is loaded to the processor and in 'Main' points to the first line of code (**Figure 16**). Alternatively by using F5 the program will start to run immediately after it has been uploaded; an LED flashes continually until the program is halted.
In the main menu under '*Debug*' you can jump to the next breakpoint and have access to all the usual debug functions.

One final tip: When the system stops responding it sometimes helps if you carry out a complete erase ('Erase Chip' option in the 'Memories' tab in Device Programming). This sometimes occurs especially if there is a previously installed program in memory.



In conclusion

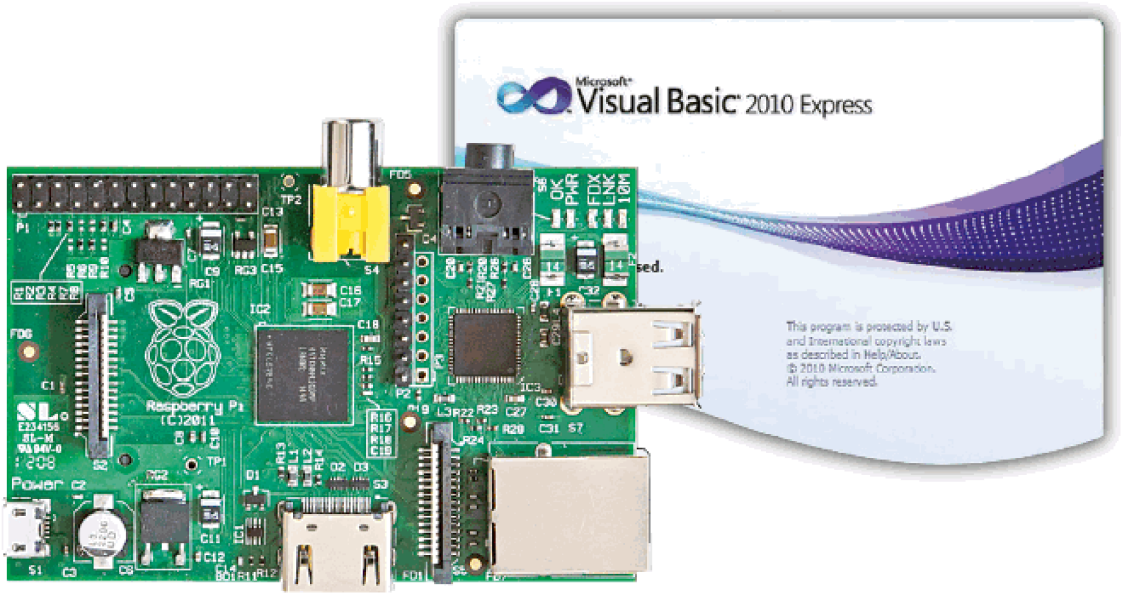
For a reasonably modest outlay the Atmel-ICE gives you the functionality of both the AVR- and SAM- debugger but don't forget the speed tradeoff. For small to medium sized projects the lower speed is hardly noticeable but when you are using larger files and working in a more intense software development environment the AVROne! with a SAM-ICE still has the edge.

(140275)

Web Links

[1] www.ineltek.de/en/index.php
[2] www.atmel.com/webdoc/atmelice/index.html
[3] www.atmel.com/tools/atmelstudio.aspx

Visual Basic on the Raspberry Pi



By **Bert van Dam**
(Netherlands)

The Raspberry Pi is normally programmed in Python. This is a powerful and easy to use language, but if you are more familiar with a language such as Visual Basic then it is not all that straightforward to make nice graphical user-interfaces using Python.

As an intermediate solution you could use a graphical template (as described in [1]), but it would be much more convenient, of course, to develop a program on a Windows PC in, for example, Visual Basic and then use it on a Raspberry Pi. In this way you can utilize the powerful computing abilities of your PC and the extensive graphical development environment provided by Visual Basic!

Visual Basic on a PC uses .NET. This is a kind of common programming framework which contains a large collection of libraries that are available for use by programmers (and applications). Programs are compiled to an intermediate code, which uses .NET. This is called the Common Language Interface. In this way programs developed for one computer type can also be used on another one, provided it also contains .NET. Dot NET is developed by Microsoft and is mainly intended for use on Windows computers, but there are also variants for Linux available, of which Mono is the most widely used.

In this article we install the Linux variant of .NET on a Raspberry Pi and, as a demonstration, run a Visual Basic program that was developed on a PC (Windows 7, 64-bit).

What do you have to do?

If you do not already have Visual Basic installed on your PC then download Visual Basic Express 2010 from the Microsoft website [2]. There is also a newer version (2013) available, but because the Linux version of .NET always runs a little behind, it is better to use the Visual Basic version of 2010. Write a test program in Microsoft Visual Basic. In the download for this article [3] you will find the source code for a simple program with one button, which is named 'monotest'. When you

run the program you will see a window with a button. When you click the button the text "Hello World" will appear (see **Figure 1**). For those of you who don't have Visual Basic on their PC (yet), the download also contains the compiled version of this program (monotest.exe).

Ensure that your Raspberry Pi is connected to the internet and install the 'mono' software package with the following commands:

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

This will take a bit of time; answer all the installation questions with Y (yes).

Copy the executable (the file monotest.exe) from your PC to the Raspberry Pi. This is very easy when using the program WinSCP, as is used in the book 'Raspberry Pi'. This program is part of the download for this article and does not need to be installed. Run the program (WinSCP.exe) and under 'Session' at 'Host name' enter the IP-address of your Raspberry Pi. Make sure that port 22 is selected. Now make the connection. When you use an IP-address for the first time in WinSCP you will see a warning; accept it. Subsequently go to 'Commander'. You will see an overview of the files on your PC (on the left) and on the Raspberry Pi (on the right). Find the executable file on the left and drag it to the RPi. Run the program on the Raspberry Pi with the command `mono monotest.exe`

You can now experiment to your heart's content with this nice and easy method. Because you are developing the program on a PC, Raspberry Pi specific components (such as GPIO pins) are not easily accessed. This method is more appropriate for making nice graphical programs, such as games, in an easy manner.

Software

The demo program ' monotest' , the accompanying Visual Basic 2010 source code as well as the program WinSCP are available as a download from the Elektor magazine website [3]. This method has been tested with the SD-card that accompanies the Raspberry Pi book, but it should also work with the standard Raspbian Wheezy SD card.

(140263-I)

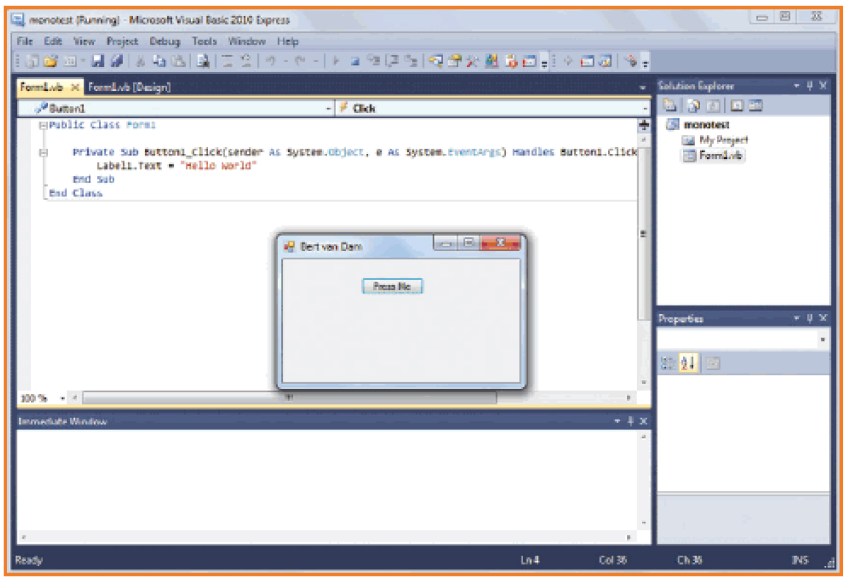


Figure 1. The program Monotest, opened in Visual Basic on a Windows PC.

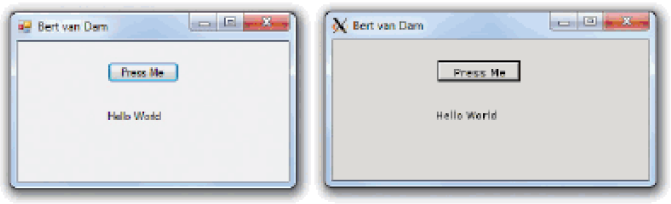
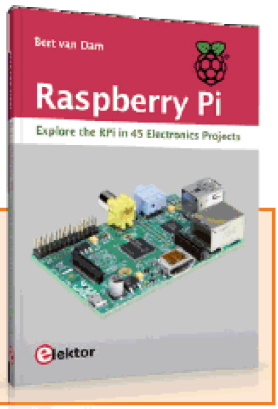


Figure 2. On the left the program on a Windows PC, on the right on a Raspberry Pi.

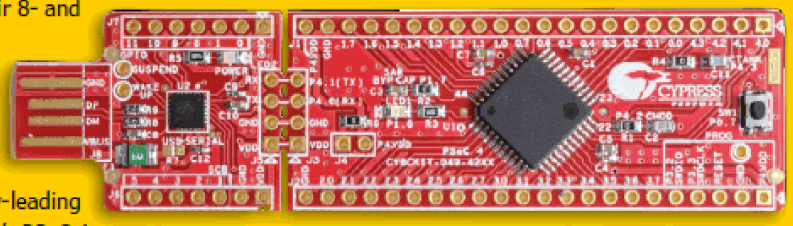
Web Links

[1] www.elektor.com/raspberry-pi
[2] http://www.visualstudio.com/en-us/downloads#d-2010-express
[3] www.elektor-magazine.com/140263



Other interesting Raspberry Pi projects can be found in the Bert van Dam authored Elektor book
'Raspberry Pi – Explore the RPi in 45 electronics projects' [1].

Upgrade legacy 8- and 16-bit designs with Cypress $4 PSoC 4 Prototyping Kit



Embedded system design engineers looking to upgrade their 8- and 16-bit legacy designs to a 32-bit ARM CPU can do so for less than a pack of AA batteries, thanks to a new PSoC® 4 prototyping kit from Cypress Semiconductor Corp. The $4 CY8CKIT-049 Prototyping Kit enables designers to leverage the powerful 32-bit ARM® Cortex™-M0 core, programmable analog and digital peripherals, and the industry-leading CapSense® capacitive touch-sensing technology of Cypress's PSoC 4 architecture. The kit can be used for both prototyping and production and comes with Cypress's free, easy-to-use PSoC Creator™ Integrated Design Environment (IDE), which features hundreds of example projects to enable rapid product development.

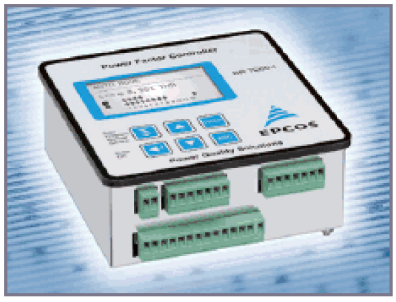
"Cypress has shattered the barrier for entry into 32-bit ARM designs with the most compelling combination of price and performance available in the market," said John Weil, Senior Director of PSoC Marketing and Applications at Cypress. "Our new $4 prototyping kit delivers the industry's most mixed-signal technology per square inch compared to competing boards. Designers can use it to quickly create prototypes leveraging our custom hardware programmability with PSoC Creator."

The 3.59-inch by 0.95-inch CY8CKIT-049 Prototyping Kit provides access to all of the I/Os on these PSoC 4 devices, allowing for fast, simple programming via an on-board bootloader through Cypress's USB-Serial Bridge Controller with a unique snap-away design. The PSoC Creator IDE complements the kit and silicon with more than 100 pre-verified, production-ready components—free embedded ICs represented by an icon—that can be dragged and dropped into designs. PSoC 4 is Cypress's newest ARM-based PSoC architecture, featuring the low-power Cortex-M0 core combined with PSoC's unique programmable mixed-signal hardware IP. The result is the industry's most flexible and scalable low-power mixed-signal architecture. PSoC 4200 devices feature a 12-bit SAR ADC, two opamps, two comparators, four Timer/Counter/PWM blocks, two serial communication blocks, and four PLD-enabled Universal Digital Blocks. Additionally, all PSoC 4 devices offer a dedicated CapSense block for fast implementation of sleek, reliable touch-sensing user interfaces. Designers can add CapSense touch-sensing to the CY8CKIT-049 Prototyping Kit without any external components by simply adding copper tape for conductivity. PSoC 4200 devices provide up to 32 KB Flash memory and 4 KB of SRAM, and they are available in 40-QFN, 28-SSOP and 44-QFP packages.

The PSoC 4 Prototyping Kit is available via Cypress's Webstore (link below) and through all franchised distributors worldwide. Visit the PSoC 4 Prototyping Kit webpage for the board schematics, example projects and the user guide.

www.cypress.com/store **www.cypress.com/?rID=92146**

Control up to 32 Thyristor Modules via RS485



TDK Corporation has extended its BR7000 series of EPCOS power factor controllers with two new types. The BR7000-I-TH controller offers 12 relay outputs for capacitor contactors and 12 transistor outputs for thyristor modules. The BR7000-I-TH/S485 features an additional RS485 bus interface that allows up to another 32 EPCOS TSM-LC-S thyristor modules to be controlled. This bus interface also enables bidirectional communication with the thyristor modules.

The new controllers are particularly well matched to the thyristor modules of the new TSM-LC-S series for dynamic power factor correction with a rating of up to 55 kvar. They detect and store key grid and capacitor parameters. This enables the implementation of complex PFC installations, which are also self-monitoring. This improves system protection and helps to increase the operating life of the capacitors. Features and benefits of the new devices include:

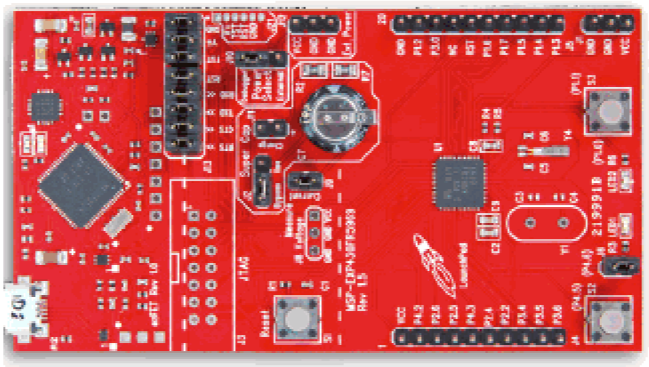
- Controlling up to 32 EPCOS TSM-LC-S thyristor modules via an RS485 interface;
- Bidirectional communication with the thyristor modules;
- Display of a wide range of parameters including the harmonics.

Both controllers already offer 20 pre-installed control series. In addition to the most important grid parameters such as voltage, current, frequency as well as reactive, apparent and effective power, they also measure the distortion of current and voltage (THD-I/THD-V). The graphics display the results up to the 33rd harmonic. These controllers are designed for voltages of between 30 V AC and 440 V AC (L-N), or of 50 V AC to 760 V AC (L-L).

Ultra-low Power FRAM with Texas Instruments LaunchPad Dev Kit

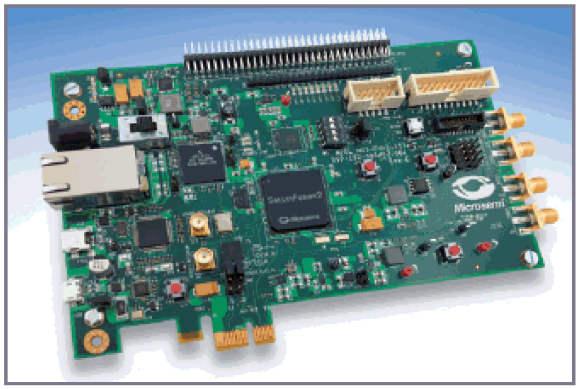
Farnell element14 has announced the launch of the Texas Instruments MSP430 ULP (ultra low power) FRAM LaunchPad, a development platform which combines embedded FRAM (Ferroelectric Random Access Memory) and a holistic ultra low-power system architecture. The new addition to the LaunchPad family, MSP-EXP430FR5969—an easy-to-use Evaluation Module for the MSP4305969 microcontroller enables developers to reduce energy budgets, minimize product size and enable a battery-free world. Embedded FRAM, a non-volatile memory known for high endurance and high speed write access, together with ultra-low power makes the MSP430 suited for a wide variety of applications ranging from metering, wearable electronics, consumer electronics, the Internet of Things (IoT), industrial and remote sensors, home automation and energy harvesting.

Available in this dev kit, TI's new EnergyTrace++ technology is the world's first debug system that enables developers to analyze power consumption down to 5 nA resolution in real-time for each peripheral; this allows engineers to take control of their power budget and optimize software to create the lowest energy product possible.



Features include:

- MSP430 ULP FRAM technology-based 16-bit MSP430FR5969 MCU
- 64KB FRAM / 2KB SRAM
- 16-Bit RISC Architecture up to 8-MHz FRAM access / 16-MHz system clock speed
- 5x Timer Blocks
- Analog: 16Ch 12-Bit differential ADC, 16Ch Comparator
- Digital: AES256, CRC, DMA, HW MPY32
- 20 pin LaunchPad standard leveraging the BoosterPack ecosystem



SmartFusion2 SoC FPGA Evaluation Kit

Microsemi Corporation announced the availability of the company's new leading-edge SmartFusion®2 SoC FPGA Evaluation Kit. The new SmartFusion2 Evaluation Kit is an easy-to-use, feature-rich, affordable platform designed to enable designers to quickly and easily accelerate evaluation or prototype their application. Utilizing Microsemi's mainstream SmartFusion2 FPGAs enables original equipment manufacturers (OEMs) to leverage the device's lowest power consumption in its class, high reliability capabilities and best-in-class security technology to build highly differentiated products that help them gain a significant time to market advantage.

A prime example is that the SmartFusion2 Evaluation Kit allows for simplified development of transceiver I/O-based FPGA designs necessary in today's PCI Express (PCIe) and Gigabit Ethernet-based systems. For faster evaluation and prototyping, Microsemi's leading-edge evaluation board is small form-factor PCIe compliant, which can be used on any desktop PC or laptop with a PCIe slot.

The kit offers a comprehensive set of features that include PCIe, Gigabit Ethernet, full-duplex SERDES SMA pairs, DDR memory, SPI Flash, USB On-The-Go and several expansion interfaces that create the needed flexibility for a wide range of application development. With purchase of the evaluation kit, developers also have access to Microsemi's full array of industry leading development resources such as reference designs and the ability to launch example application demonstrations.

www.microsemi.com/fpgaevaluationkit (140301-IV)

Parallax Propeller Released as Open Source Design

Parallax Inc. has released their source code design files for the Propeller 1 (P8X32A) multicore microcontroller among the 13,000+ attendees of the DEF CON 22 Conference in Las Vegas where their chip is also featured on the conference's electronic badge. Parallax has long-believed in openly sharing product designs for the benefit of its users and the development community. CEO Ken Gracey anticipates this release will inspire developers the same way Parallax founder Chip Gracey was motivated when he taught himself to program computer systems in the early 1980s. Hobbyists, engineers, and students may now view and modify the Propeller Verilog design files by loading them into low-cost field programmable gate array (FPGA) development boards. The design was released under the GNU General Public License v3.0.

With the chip's source code being available, any developer may discover what they need to know about the design. The open release provides a way for developers who have requested more pins, memory or other architectural improvements to make their own version to run on an FPGA. Universities who have requested access to the design files for their engineering programs will now have them.The Propeller multicore microcontroller is used in developing technologies where multiple sensors, user interface systems, and output devices such as motors must be managed simultaneously. Some primary applications for Parallax's chip include flight controllers in unmanned aerial vehicles, 3D printing, solar monitoring systems, environmental data collection, theatrical lighting and sound control, and medical devices.

The decision to go open-source has positive implications for Parallax's next design: the Propeller 2. In recent years the community contributed the compilers, multi-platform programming tools and languages for the Propeller 1. Several of the Propeller 2 features were contributed and designed by community members running binary images of the chip in an FPGA. Though a small supplier in the semiconductor industry, the high quality of Parallax's products can be attributed, in part, to their close relationship with the development community.

www.parallax.com (140335-I)

A 1965 Telefunken Carphone

"I'm running a little late, Schätzchen"

By Gerd Kowalewski
(Germany)

Reproduced courtesy
Museum für Kommunikation Frankfurt.

Today we can grab a slim, lightweight mobile handset and use it to phone people anywhere across the globe. To satisfy our communication and information needs we carry smartphones, optimized for energy saving, that not infrequently feature multiple microprocessors and a high-resolution color display in order to go online using the powerful infrastructure of modern cellular mobile networks.

Figure 1.
The garage find: two identical sized, metal, trunk-style cases marked Telefunken on what appear to be front panels.

Figure 2.
The twin equipment cases of the mobile telephone setup, together with the cable assembly

Today these little electronic companions perform far more than the simple voice communication of phones made in days gone by. We communicate not only the signal of our digitized speech but also the written word of e-mails and text messages. We also share data files, music and photos. What's more, we can network multimedia, obtain geodata for navigation and view videos from files of highly-compressed audiovisual content. We engage in live streams to have virtual presence at any event in the world, making the most of the unlimited possibilities of the World Wide Web.

A standard mobile handset connected to the cellular mobile networks type D, E up to LTE has a performance that surpasses previous desktop computers by miles—using a fraction of the energy requirements. Today we can thank technological progress in general and microelectronics in particular for these lightweight, portable gadgets with their substantial energy capacity providing us with communication capability for days on end before they need to have their energy replenished in a charging device.

In the process we frequently overlook the fact that the essential prerequisite for all this is 'the

clever technology behind the scenes', the infrastructure systems of the mobile radio networks.

Garage find

The discovery of a VHF personal mobile radio (PMR) device that today is very rare (on account of the comparatively small number made) aroused the author's curiosity and provided the motivation for taking a backwards look at the origins of the history of mobile radio networks in his country, Germany.

In the estate of a deceased person, carefully packed in a dusty cardboard box and stored for decades, were found two rather unprepossessing and totally unspecific sheet-steel cases, on which the name of the renowned German manufacturer Telefunken was resplendent (**Figure 1**).

Of course this alone was enough to arouse the curiosity of an electronics enthusiast. Further research indicated that this discovery was a very special treasure: an in-car transmitter-receiver type S/E 160E15 öbl B plus power supply, selective calling (selcall) unit and even the complete cable harness (**Figure 2**), from the earliest days of the first German public mobile radio system, the A-Network of around 1960, in an excellent state of preservation. The interconnections between the two cases are pictured in **Figures 3a, 3b** and **4a, 4b**.

The suffix 'öbL B' on the designation plate and the SO-239 "UHF" connector provided the definitive evidence to the purpose of this equipment. The type designation of the equipment 'S/E160E15' tells a lot. First, it's a receiver/transmitter (Sender/Empfänger; T/R), second, the radio frequency band around 160 MHz and third, 15 selectable communication channels out of a total of 17 possible in the A1 Network of that period. Two VHF carrier frequencies, offset by 4.5 MHz, created a duplex radio channel with support from base stations in elevated locations about 50 kms (35 miles) apart. This device, today extremely rare, was originally designed in the city of Ulm by a radio specialist company Telefunken, as a significantly more compact successor to the S/E 160E11. Production, in relatively small numbers and assembled mainly by hand, began in 1965 (as the official approval number indicates).

Initially only a handful of mainly very affluent customers used the A-Network [1] of the public mobile land radio service (*öffentlicher beweglicher Landfunkdienst* or öbL) governed by the German Post Office. The capacity limit of its final

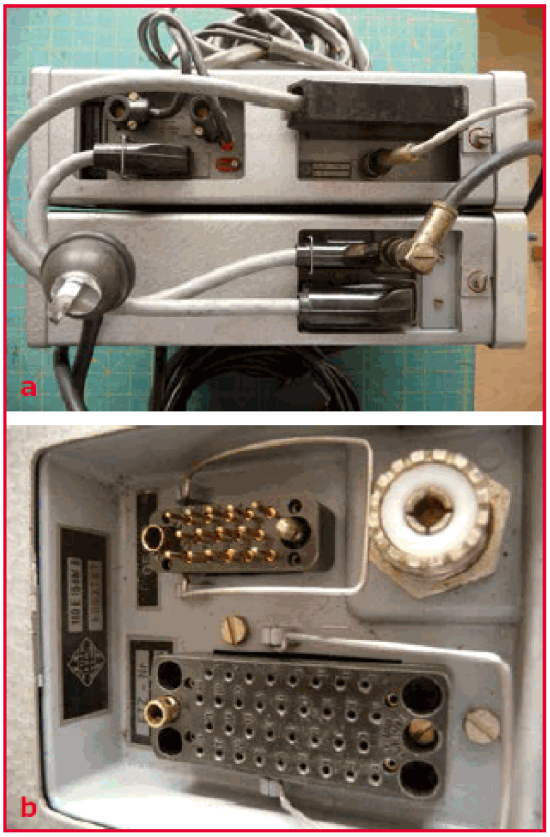


Figure 3.
(3a) Cable connections between the S/E 169E15 öbL B radio box and the ancillaries box. (3b) View of the connector bay on the radio box.

Figure 4.
Views of the connector area on the power supply unit STV E 6/12 (4a) and the selcall unit SRS-1098/12 (4b). *Ruf-Nr* = selcall ident.

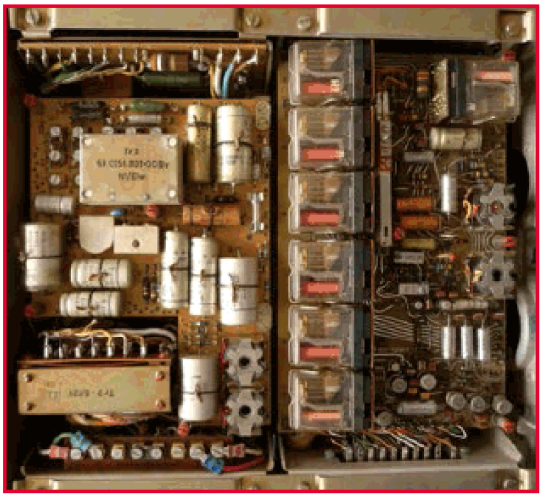


Figure 5.
This is what the push-pull DC-DC converter STV E 6/12 and selcall unit SRS-1098/12 look like.

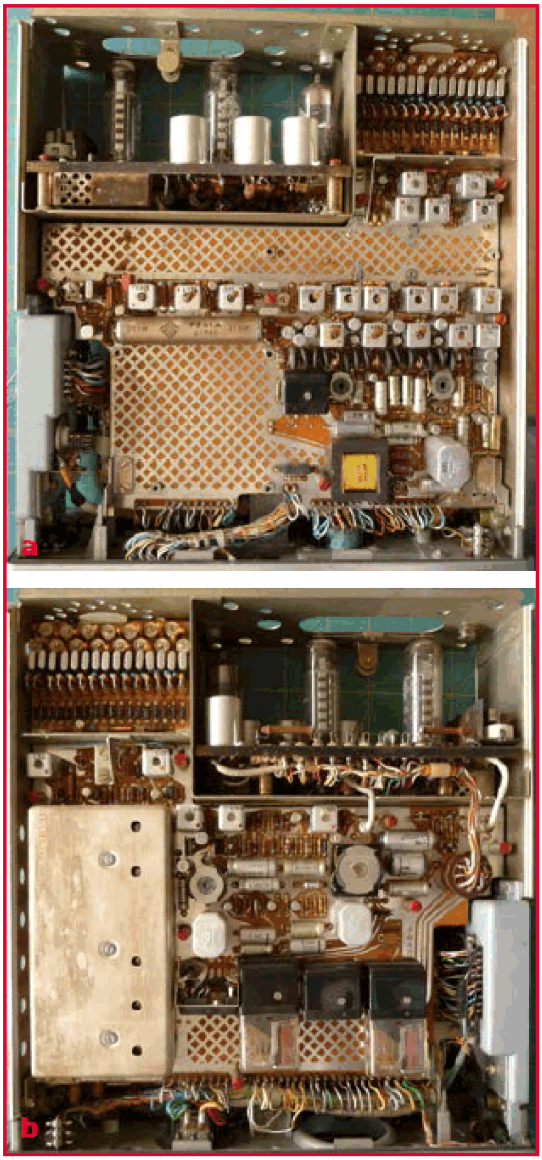


Figure 6.
Internal views of the transceiver. It's All-Transistor—well nearly!

incarnation was reached in 1971 with the then barely imaginable number of 10,784 subscribers. This is surely one reason for the current rarity of car mobile phones of this kind. Yet there was also another, equally realistic reason: a car mobile telephone setup comprising two equipment cases for trunk mounting and a control unit on the dashboard cost a fortune!

This high-tech (for the time) device weighed in at around 30 lb (15 kg) and was stated to consume only (!) 24 watts (2 A fuse) [to be verified, *Ed.*]. It was already transistorized including the push-pull switching HV converter in the second equipment case (**Figure 5**), but excepting the multiplier, driver and 10-watt PA tube in the transmitter, two of which were double tetrodes type 6360 (QQE03/12) (spot them in **Figure 6**). Some jumpering had to be done to match the kit to the electrical system on the vehicle (**Figure 7**). No, not everyone was on 12-volts negative chassis in the 1960s.

The equipment would have cost a trifling DM 6,550 excluding installation charges! [6]. This sum, equivalent to $33,600 / £19,630 in today's money, would have enhanced the value of even a luxury 'senior management' limousine significantly at the time.

A purchase of this kind (not to mention its running and license costs) lay beyond the means of a German salaried worker, with monthly standing charges alone amounting to DM 65 ($350 or £205 at current values). Later the authorities fended off the onslaught of everyone interested with charges of DM 270 ($1,390 or £810).

Clients of the mobile radio telephone service included not only politicians and well-off business people but also road maintenance and forestry commission teams, ground crews at airports and the skippers of ships traversing the inland waterways of Germany. The German Federal Railways also exploited the network for their radiophone service aboard trains.

On the A1-Network one channel was sufficient for around 20 to 25 subscribers, who could all cheerfully hear any other user's calls. Obviously this had to change. The A2-Network added 19 duplex channels (38 carrier frequencies). Channel 39 was defined as the calling channel. A 2280-Hz 'channel clear for use' tone signifying to users that they were still within radio range and connected to the fixed telephone network, was transmitted by local radio base stations, together with the 1750 Hz frequency indicating that allocated channels were occupied. These tone frequencies were transmitted in the

communications channel. Without the 'clear' (= 'not in use') tone from the base station a speech channel could not be assigned ('not in range').

Selective calling

Underpinning the 7-digit call number [2],[3] of a mobile subscriber lay a relatively complex coding scheme. The first and second digits represented the so-called Subscriber Key determining the zones within the Radio Network Maps, either for the A1-Network (digits 21-25, 31) or the A2-Network (digits 61-65, 32). The established idea of audio frequency signaling in the calling channel was integrated into a system for coded call signaling by means of a special scheme employing precise (1% tolerance) audio frequencies. The so-called Group Key [3] was encoded in the third digit of the mobile number. These digits figured in the '4 out of 30' selective calling system according to a special key table in which four individual frequencies were chosen from a total of 30 audio frequencies that stood for a four-digit ordinal number (digits 4 to 7 of the mobile radiophone number). This technique enabled a so-called 'continuous ringing signal' to be used on the calling channel for semi-automated coded calling of a maximum of 27,405 subscribers, assuming the client's location is known at least roughly in advance. No roaming, triangulation, cell handover in those days.

All this demanded extremely narrowband tuned circuits in the receiver that had to detect, simultaneously, four audio frequencies on the service channel (supervisory channel in telephone-speak), in a scheme using audio frequencies with f(n)= 337.5 Hz + n×15 Hz where n= 1... 30. Initially n= 20 sufficed for the number of code combinations. For the A3-Network the selective calling system was expanded to n= 30.

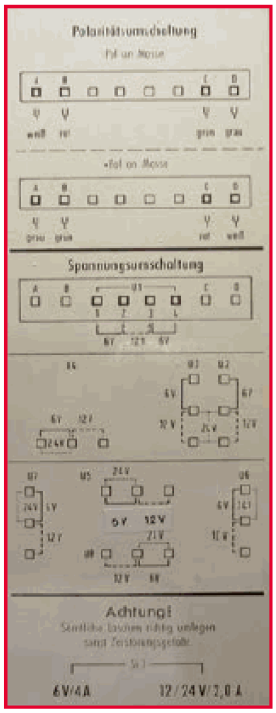


Figure 7. Yes please? 6 volts, 12 volts or 24 volts DC operation? Positive or negative on chassis? Just roll in that VW Microbus!

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

What's in the boxes and...

Equipment case 1 contains the VHF radiotelephony transmitter-receiver 160E15 öbL in the B variant, type G 50-3. As such it includes all the actual communications technology of a remote-controlled system (by cables) with the following specifications:

15 channels, out of a total of 17 duplex channels on the A1-Network, thus 34 VHF carrier frequencies, crystal-controlled (>30 crystals), in this instance equipped with channels.

- CH30: 157.55 MHz / 162.05 MHz to CH44: 158.25 MHz / 162.75 MHz.
- 4.5 MHz offset for full duplex operation with 50 kHz spacing.
- Operating mode: FM (F3); 4 kHz deviation.
- Handshake tone: 1750 Hz ±1 Hz.
- RF transmit power: 10 W [before duplexer, *Ed.*]
- Antenna connector: Amphenol SO-239.

Equipment case 2 contains the STV / SRS equipment set, with the STV E 6/12 assembly, namely power conditioning from a 6 V/12 V/24 V vehicle supply with either positive or negative ground connection to chassis, using a fully transistorized DC-DC converter to generate and stabilize all of the operating voltages.

Power consumption at 12.6-V vehicle battery voltage: 24 W (2 A fused for 12/24V).

Also incorporated is the selcall unit SRS 1098/12 for frequency call signaling over the service channel.

History of analog mobile radio in Germany

As long ago as 1918 the first experiments with radio telephony began on the German railways, initially on Long Wave and later on the so-called 'fringe wavelengths' in the region of 30 MHz. Beginning in 1926, passengers aboard express trains on the Berlin-Hamburg route could make radio telephone calls on a regular basis, and six years later, in 1932, the Mitropa service company connected telephone calls between moving trains and ships at sea using marine radio. Consequently mobile radio telephony was already well developed before the seizure of power by the Nazis in 1933 but these services were later suspended as a result of the war.

In postwar Germany, initially during the occupation by the victorious Allied powers and the period of demilitarization, demobilization and denazification, the situation began to alter in the three western occupation zones of Germany following the American Marshall Plan (1947), currency reform (1948) and the first tough years of reconstruction. In 1949 Western Germany received its democratic constitution and its division into federal states.

The federal German Post Office administration acquired the rights covering the use of radio according to international agreements and previous restrictions and sanctions of the victorious Allied powers.

Starting in 1958, the postal administration, in those days still the supreme authority over all telecommunications in the country, consolidated the disparate radio services of the developing economic wonderland into the so-called 'A-Network' (A-Netz) in the frequency band from 156 to 174 MHz to create a 'public land mobile radio service' (*öffentlicher beweglicher Landfunkdienst*, öbL for short). This system was expanded further in stages termed A1, A2 and A3. By 1968 coverage had reached 80 % of West Germany.

Roll-out of a more efficient mobile network began in 1972 with the B1-Network of 38 channels. Subscriber numbers continued to increase strongly up to the end of the 1970s, despite comparably higher charges, so that under the constraints of the permanently limited number of spare radio channels, it was necessary to 'recycle' the carrier frequencies of the A-Network to use on the new B2-Network, achieving in 1980 a total of 75 speech channels. In the process the new channel spacing shrank from 50 kHz to 20 kHz, with Channel 19 becoming the standard calling channel throughout the federal states.

Thanks to multiple re-use of these channel frequencies, in radio cells that were spaced adequately apart from one another, the German Post Office effectively had 850 frequencies for simultaneous use to accessing the fixed telephone network. The B-Network also made it possible for the first time to make a call without the need for manual connection by the 'switchboard girl' – so long as mobile subscribers could establish on a map in which of the 158 different radio zones in the federal republic they were, along with the appropriate dialing prefix for reaching the fixed network. 'Roaming' across state boundaries was not yet possible in those days!

The battery cable, the original connecting wires and even the old antenna coax cable complete with antenna base (without the whip antenna, however) are still present after 50 years!
Physical dimensions of each 'trunk-box' style case: approx. 305 mm long x 270 mm deep x 85 mm tall. Total weight: 11.6 kg, including cable harness. Recommended in-line fusing for complete equipment: 15 A.

Figure 8.
The standard AT400-series "Becker" control box type for dashboard mounting. Still a bit unpolished! (courtesy and copyright oebl.de)

... what's not

Unfortunately the Control Box, typically one from the Becker AT400 set [4], is missing. Most likely it disappeared along with the vehicle in which it was fitted. The box (**Figure 8**) consisted merely of a couple of signal lamps, a buzzer and some switches, finished off with a 'Funk 70' telephone handset [5]. There was no need for any more, since the main comms functionality was contained inside the two apparatus cases.
The control box alerted the user of an incoming call from the radio exchange by means of signal lamps and a buzzer. Additionally a green light showed 'system available' when a call could be made to the nearest radio exchange. A rotary switch also made it possible (in conjunction with a map of radio zones) to select the correct radio channel for the locality according to the user's location and the radio zone within the Federal Republic. Better finished control boxes appeared on the market later (**Figure 9**), as wells "auto-

By 1986 the continually booming growth in subscriber numbers was pushing the B-Network, with around 27,000 users, to its limits; the German Post Office had to order a block on further enrolment!
Incidentally, the first handheld set, Motorola's Dynatac 8000 (known in Germany as 'The Bone', with around 30 minutes operating time) was introduced on 13 June 1983, in other words more than 30 years ago!
From 1 September 1985 the C-Network provided an alternative for mobile telephony in the 450 MHz band. This was a 'first', offering fully automatic switching using the single access code 0161 across the federal republic. Also new was 'handover', the ability to continue a conversation in motion from one radio zone to the next, with it now being possible to determine the user's location within the radio network automatically. The B-Network was not taken out of operation until 1994.
In its glory days the C-Network had around 800,000 users. Over time the monthly standing charges fell by 84 percent, whilst the radio equipment became significantly smaller and lighter, making the development of portable handsets possible for the first time. The C-Network persisted up until its switch-off in 2000.
Today we term all the A, B and C-Networks together as first-generation (1st Gen. or '1G') mobile systems. Whilst the equipment of the first generation(s) still represented genuine speech devices with analog voice signal transmission, a new digital technology soon appeared on the horizon.

Information Sources

(in German but Google Chrome will translate these pages):

http://izmf.de
www.oebl.de/A-Netz/ANetz.html
www.oebl.de/B-Netz/BNetz.html
www.oebl.de/C-Netz/CNetz.html
www.mobilfunk-geschichte.de
http://wissen.de/die-geschichte-der-mobiltelefone
http://de.wikipedia.org/wiki/A-Netz
http://de.wikipedia.org/wiki/B-Netz
http://de.wikipedia.org/wiki/C-Netz
http://de.wikipedia.org/wiki/D-Netz
http://de.wikipedia.org/wiki/E-Netz
www.3gpp.org
www.bundesnetzagentur.de/DE/Sachgebiete/Telekommunikation/Unternehmen_Institutionen/Marktbeobachtung/Deutschland/Mobilfunkteilnehmer/Mobilfunkteilnehmer_node.html
http://en.wikipedia.org/wiki/Universal_Mobile_Telecommunications_System
http://en.wikipedia.org/wiki/LTE_(telecommunication)
http://en.wikipedia.org/wiki/LTE_Advanced
www.focus.de/digital/handy/mobilfunkgeschichte/

The Author

In his youth, Gerd Kowalewski started collecting and repairing tubed TV sets and other stuff. Later, slightly educated, he studied Electrical Engineering and joined the start-up company CPV initially designing datacomms equipment, and eventually handling Product Development & Far Eastern Productions. He spent several years in the electric vehicle (EV) business and instrumentation. Next, think-tanking Mannesmann Pilotentwicklung (*mpe*), Munich, resulted in some patents. After running his own mini business GK Electronic Consulting (*GKEC*) for 10 years, health reasons forced an early retirement. Today, having read Elektor magazine for 40 years Gerd has a continued interest in tricky circuit designs and efficient HW solutions—as well as in all scientific advances, future or past.

matic" ones **(Figure 10).** Besides an indication of radio fieldstrength some really deluxe versions also showed the voltage of the vehicle battery, as so many people forgot to switch off the radio set when leaving the car. The speech apparatus was a telephone handset very similar to that used with the standard 'W48' instrument used widely on the landline network [5]. The handset, together with the control box, was normally attached to the vehicle's dashboard next to the car radio (or else beside the VIP in the back of the car).

Control boxes of this kind could be had not only from Telefunken but also from other suppliers, such as the German firm Becker, presumably on account of pre-existing supply arrangements between the 'premium' car manufacturers of the time (Mercedes-Benz and the former Opel AG) and the established suppliers of automobile radios (which in those days went under the name 'Autosuper').

Figure 9.
Special version of the carphone control box for the Mercedes W108/W109 dashboard. Merc fans in California, Texas, Florida, Nevada, eat your hearts out. (courtesy and copyright oebl.de)

Figure 10.
"Automatik" or "deluxe" versions of the control box soon superseded the standard version.

Summing up

This Telefunken S/E 160E15 öbL demonstrates clearly—particularly to members of the younger generation—how rapidly mobile radio has developed in Germany within the space of only 50 years or so.

Without the development effort of the radio networks that have constantly underpinned them, beginning with the A-Network of land mobile radio all the way to today's broadband systems of the current Long Term Evolution (LTE), each mobile radio telephone would amount to nothing more than a collection of electronic components. Truly a long-standing evolution!

The equipment I was fortunate to find appears to be capable of operation but has not been tested so far, as a competent restorer would undoubtedly first change some capacitors, either reforming or replacing them, before even powering up anything, in order to avoid causing any damage. A technical museum, collector or restorer can probably start on the equipment using the information collated here. The system was put up for sale on Ebay and meanwhile found a good home with a private collector.

(140268)

Information Sources

(in German but Google Chrome will translate these pages):

[1] www.oebl.de (a remarkable private collection assembled by Stephan Hessberger)

[2] www.oebl.de/A-Netz/Technik/Technik.html

[3] www.oebl.de/A-Netz/Technik/Tabelle.html

[4] www.oebl.de/A-Netz/Geraete/becker/AT400/AT400.html

[5] www.oebl.de/sonstiges/Hoerer/hoerer.html

[6] www.oebl.de/A-Netz/Doku/AT400_Angebot.JPG

Knowing *vs* Understanding

By Gerard Fonte (USA)

One of my favorite sayings is: "Do you believe? Or do you understand? If you believe then you let the word-givers have power over you. If you understand then no one is your master." Believing, or knowing, certainly has its place. But it's important to realize that believing can blind you to alternatives.

No Time to Think

Believing allows us to act quickly. We don't have to stop and analyze the situation before doing something. I know right away what an angry dog looks like and immediately take steps to avoid an attack. I don't say to myself: "The dog's teeth are bared, his ears are flat against his head, he's growling, his tail is not wagging, he's staring at me and advancing. These are indications that the dog is likely to attack so I should do something." Obviously, if I did that, my leg would be a chew toy.

We are taught what to know from birth. We learn from others, and from experience. Since humans have very developed language skills, most of what we know is second hand. Our personal experiences make up a relatively small amount of what we know. Consider that from birth to adulthood being educated is the most important aspect of our life. As infants we learn to walk and talk, feed and dress ourselves, and master the skills of socializing. Then we spend twelve years in school followed by four or more years of college. Only then are we expected to make our way in life by ourselves. (But that's the one thing that's not really taught at all. It's something we each learn by trial and error.)

We know and believe what we are taught. This is just human nature. And it is our nature to trust these word-givers. Learning that two plus two equals four is pretty much the standard everywhere. Being taught political or religious ideas can be somewhat more problematical. But the teaching process is basically the same for math, science, history, politics and religion. Other people tell us what to believe and what to know. And in the vast majority of situations, this is practical and beneficial. However, once you "know" something you automatically make it much more difficult to "un-know" that thing or to see it from a different perspective. Quite simply, your brain sets up roadblocks to anything that is different from what you believe.

Set Blindness

Psychologists call this phenomenon, "set". It is a powerful and insidious condition. A classic illustration of this comes from a simple experiment. Subjects were asked to solve the "three-jug problem". This is where you have three jugs that hold different volumes of water and you have to fill and empty them in a particular order to obtain some specified volume of water. (If you are unfamiliar with this puzzle, do an internet search. I'll wait.)

The psychologists repeated the exercise many times with different sizes for the jugs. However, the pattern of filling and emptying was always the same. Eventually, the subjects learned the pattern and could solve the problem immediately and without thinking. They knew the solution.

Then the psychologists (being devious by nature) changed the answer. Instead of a complicated sequence of filling and emptying, they made the solution trivial: just fill the small jug from the big jug and the remaining volume was correct. Normally this simple problem was solved virtually instantly. However, with the subjects that believed that they had the answer, this elementary task was extremely difficult to figure out. They took incredible amounts of time to find the solution. Some even gave up and claimed that it was impossible! Set simply blinded them. They believed in the pattern. They were wrong.

Curiously, the more we know, the more chances we have for set to camouflage a different approach to a problem. That's an issue for conventional education. And admittedly it's usually a rare issue. But sometimes, it can be very important to see new solutions instead of the status quo. This is especially vital when the status quo changes. But, to do that, you have to understand.

Cognito ergo sum

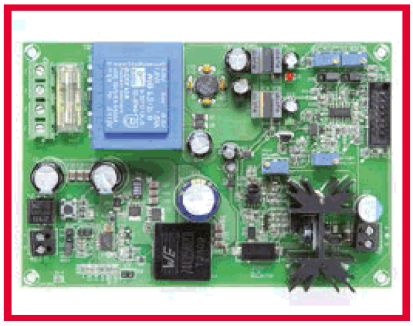
Understanding requires the use of rules and principles to examine a situation. Another name for this is thinking. It's hard to think. It takes time and effort. It's much easier to "know" what the answer is than to figure it out. But if you understand the underlying concepts you can deduce many details and facts. This means you don't have to remember all the trivia about something. You can create it whenever you need to. This removes brain-clutter and helps to organize your thinking. Instead of working with details that you know, you can work with concepts that you understand. This is a higher level of cognition and can be very effective (see March 2012, "Conceptual Engineering").

It's much easier to understand something if you do it for yourself. In this, self-learning is vastly superior to classroom learning. Self-learning is an advantage that hobbyists have and it's also called experience. Of course, the more you do for yourself, the more experience you gain. Putting together a kit requires less understanding than designing and building a circuit. But kit-building is a prelude to design and it helps understand soldering and assembly techniques. All experience is beneficial, regardless of the level.

Concluding, the more you believe the less you understand. And the more you understand the less you need to know.

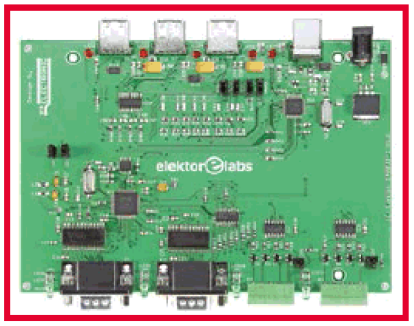
(140337)

NEXT MONTH IN ELEKTOR MAGAZINE



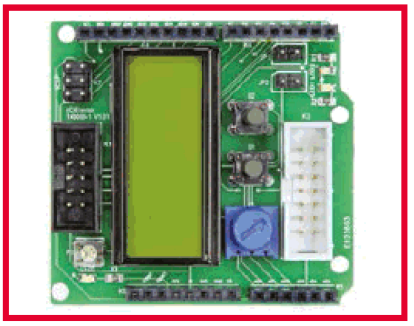
Universal 40 V/2 A Lab Supply

Several lab supplies were published in recent editions, each with its own merits and properties. In the November edition we'll describe Elektor Labs' proprietary approach—a microcontroller-controlled circuit that uses both switching and analog sections to combine in a single design essential points such as immaculate output voltage, optimal line response and low dissipation behavior.



USB Hub with Legacy RS232 and RS422/485 *

Electronics designers often run into a problem with modern computers no longer having the legacy serial interfaces, while many microcontroller circuits rely on them for communication. This handy circuit offers a universal solution: it contains a USB hub with three USB connections, and in addition has two full duplex RS232 and two RS422/485 ports.



C Software Modules

The experimental Shield presented in the July & August 2014 edition provides a good basis for all kinds of experiments based on an Arduino. Especially for this Shield we developed C-code based on the EFL (Elektor Firmware Library). For each hardware module on the Shield there's a matching software module—all you have to do is merge the desired blocks to get a functioning prototype.

** due to space constraints this article could not be published in the current edition.*

Article titles and magazine contents subject to change, please check www.elektor-magazine.com for updates.

Elektor's November 2014 edition is processed for mailing to Gold Members starting October 20, 2014.

Please note: as of the October 2014 edition Elektor magazine is not available from bookshops, newsstands and kiosks. Readers not having an Elektor membership can purchase printed or digital copies of individual magazines directly from the publishers at www.elektor.com (click on MAGAZINES).